

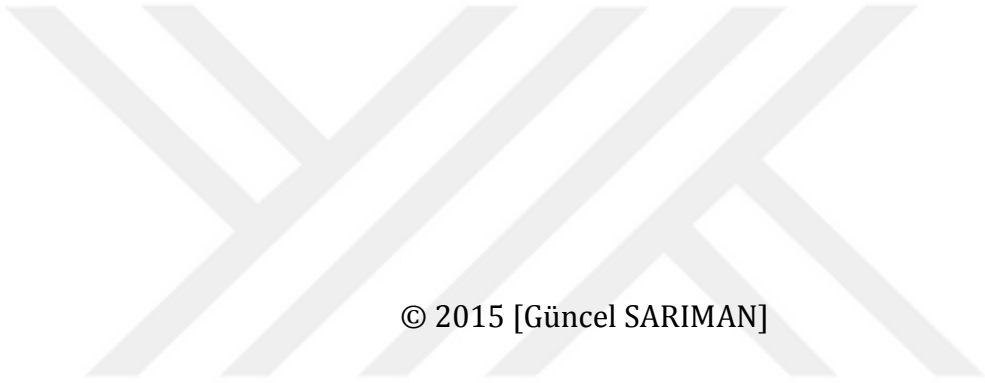
**T.C.
SÜLEYMAN DEMİREL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ**

**YAZILIM GÜVENLİĞİ TEST VE DEĞERLENDİRME ARACI
GELİŞTİRİLMESİ**

Güncel SARIMAN

**Danışman
Doç. Dr. Ecir Uğur KÜÇÜKSİLLE**

**DOKTORA TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
ISPARTA - 2015**



© 2015 [Güncel SARIMAN]

TEZ ONAYI

Güncel SARIMAN tarafından hazırlanan "Yazılım Güvenliği Test ve Değerlendirme Aracı Geliştirilmesi" adlı tez çalışması aşağıdaki jüri üyeleri önünde Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü **Bilgisayar Mühendisliği Anabilim Dalı**'nda **DOKTORA TEZİ** olarak başarı ile savunulmuştur.

Danışman Doç. Dr. Ecir Uğur KÜÇÜKSİLE
Süleyman Demirel Üniversitesi



Jüri Üyesi Doç. Dr. Ahmet YARDIMCI
Akdeniz Üniversitesi



Jüri Üyesi Doç. Dr. Özcan ASILKAN
Akdeniz Üniversitesi



Jüri Üyesi Doç. Dr. Tuncay AYDOĞAN
Süleyman Demirel Üniversitesi



Jüri Üyesi Yrd. Doç. Dr. Kubilay TAŞDELEN
Süleyman Demirel Üniversitesi



Enstitü Müdürü Doç.Dr.Yasin TUNCER

TAAHHÜTNAME

Bu tezin akademik ve etik kurallara uygun olarak yazıldığını ve kullanılan tüm literatür bilgilerinin referans gösterilerek tezde yer aldığını beyan ederim.

Güncel SARIMAN

İÇİNDEKİLER

	Sayfa
İÇİNDEKİLER.....	i
ÖZET	iv
ABSTRACT	vi
TEŞEKKÜR.....	viii
ŞEKİLLER DİZİNİ	ix
ÇİZELGELER DİZİNİ	x
SİMGELER VE KISALTMALAR DİZİNİ.....	xii
1. GİRİŞ.....	1
2. KAYNAK ÖZETLERİ	9
3. MATERYAL VE YÖNTEM	15
3.1. Statik Kaynak Kod Analizi.....	15
3.1.1. Statik kod analiz kategorileri.....	17
3.1.1.1. Tip denetimi.....	18
3.1.1.2. Stil denetimi.....	18
3.1.1.3. Program anlama.....	19
3.1.1.4. Hata bulma	19
3.1.1.5. Güvenlik İncelemesi.....	19
3.1.2. Statik analiz yöntemleri.....	20
3.1.2.1. Tip çıkarımı.....	20
3.1.2.2. Veri akış analizi	21
3.1.2.3. Kusur analizi.....	21
3.1.2.4. Sözcüksel analiz.....	23
3.2. Web Uygulama Güvenliği	24
3.2.1. Web Uygulamalarının zayıflık tespiti	25
3.2.2. Web uygulama açıklıkları.....	27
3.2.2.1. Yetkilendirme.....	28
3.2.2.2. Şifreleme	28
3.2.2.3. Bilgi sızıntısı.....	28
3.2.2.4. Girdi doğrulama	29
3.2.3. Web uygulama saldırı tipleri.....	30
3.2.3.1. Sql Enjeksiyonu	30

3.2.3.2.	Çapraz Site Betiklemeesi.....	33
3.2.3.3.	Oturum kimliği açığı	36
3.2.3.4.	Siteler arası istek sahteciliği.....	36
3.2.3.5.	Dizin gezinimi.....	39
3.2.3.6.	Ldap enjeksiyonu.....	40
3.2.3.7.	Javascript enjeksiyonu.....	42
3.2.3.8.	Xml sorgulama dili enjeksiyonu	43
3.2.3.9.	Kod enjeksiyonu.....	44
3.2.3.10.	Bellek taşması	45
3.2.3.11.	Dos saldırıları	47
3.2.3.12.	Ddos saldırıları.....	49
3.3.	Yazılım Kod Kalitesi	51
3.3.1.	Yazılım kod metrikleri.....	51
3.4.	Naive Bayes Algoritması.....	58
3.5.	Düzenli İfadeler.....	61
3.6.	Web Ortamında Veri Gönderme Yöntemi.....	63
3.6.1.	Asp.Net teknolojisinde veri aktarma yöntemleri.....	65
3.6.2.	Php'de veri aktarma yöntemleri.....	68
3.6.3.	Java'da veri aktarım yöntemleri	70
3.7.	Yöntem	70
3.7.1.	Kaynak kod analiz yazılımı	70
3.7.2.	Yazılım güvenlik seviyesi belirleme aracı	72
4.	ARAŞTIRMA BULGULARI.....	74
4.1.	Kaynak Kod Analiz Yazılımı	74
4.1.1.	Yazılım veritabanı	74
4.1.2.	Girdi denetimi.....	75
4.1.2.1.	Asp.net teknolojisinde incelenen zafiyetler	76
4.1.2.2.	Php dilinde incelenen zafiyetler	85
4.1.2.3.	Java'da tespit edilen zafiyetler.....	90
4.1.3.	Metrik analizi	95
4.1.3.1.	Asp.net, php ve java dillerinde ölçülen metrik türleri.....	96
4.1.4.	Stil Denetimi.....	98
4.1.4.1.	Asp.net, php ve java dillerinde uygulanan stil denetimleri	99

4.2. Güvenlik Seviyesi Belirleme.....	104
4.2.1. Online güvenlik yazılımı veritabanı	104
4.2.2. Güvenlik seviyesi belirleme yazılımı	105
4.3. Bulguların Değerlendirilmesi.....	110
4.3.1. Girdi Denetimi Sonuçları	111
4.3.2. Metrik Analiz Sonuçları.....	116
4.3.3. Stil denetim sonuçları.....	118
4.3.4. Online güvenlik analiz yazılım sonuçları.....	120
5. SONUÇ VE ÖNERİLER.....	122
KAYNAKLAR	132
ÖZGEÇMİŞ.....	140

ÖZET

Doktora Tezi

YAZILIM GÜVENLİĞİ TEST VE DEĞERLENDİRME ARACI GELİŞTİRİLMESİ

Güncel SARIMAN

Süleyman Demirel Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Dalı

Danışman: Doç. Dr. Ecir Uğur KÜÇÜKSİLLE

İnternetin günlük hayatın birçok alanında kullanılmasıyla birlikte, web ve mobil uygulamaların sayısı gün geçtikçe artmaktadır. Kişisel bilgilerin paylaşıldığı uygulamalardaki güvenlik ve kullanılabilirlik ise, kullanıcılar için dikkat edilmesi gereken bir faktör olmaya başlamıştır. İnternet ortamındaki güvenlik zafiyetlerinin birçoğunun geliştirilen yazılımlarda tespit edilmesi, güvenli yazılım geliştirmenin önemini giderek arttırmaktadır. Uygulamalardaki güvenlik açıklarının henüz ürün geliştirme aşamasında tespit edilmesi ise, yazılım firmaları ve kullanıcılar için ortaya çıkabilecek zararı büyük ölçüde azaltmaktadır. Web uygulamalarındaki güvenlik açıklarının tespit edilmesi aşamasında statik ve dinamik testler kullanılmaktadır. Dinamik testler daha çok uygulamaya sızarak gerçekleştirilmektedir. Statik testler ise yazılımın byte veya derlenmemiş kodları üzerinde yapılmaktadır.

Bu çalışmada, web uygulamalarındaki güvenlik zafiyetleri tespit edilerek programcılara, güvenlik açığı barındıran kod satırları hakkında bilgi verilmesi amaçlanmıştır. Çalışma kapsamında asp.net, php ve java dillerinde geliştirilen web uygulamalarındaki zafiyetleri tespit edebilmesi amacıyla, test ve değerlendirme yazılımı geliştirilmiştir. Yazılım, derlenmemiş kodları statik kod analiz yöntemiyle girdi denetimi, metrik analizi ve stil denetimlerine göre değerlendirmektedir. Geliştirilen yazılımın geçerliliğini test etmek için asp.net, php ve java da açık kaynak olarak sunulan ve aktif olarak kullanılan 5'er adet web projesi değerlendirilmiştir. Girdi denetimiyle uygulamalardaki satırlar zafiyetlerin türüne göre taranarak kullanıcılara detaylı rapor verilmiştir. Sql, Header, XPath Enjeksiyonu, XSS, CSRF, File Inclusion, Config tiplerinde tarama yapan yazılım elde ettiği sonuçları kullanıcıya grafiksel rapor ve detay olarak vermektedir. Yazılım, metrik analizi ile projelerin kaynak kod kalitesi hakkında bilgi vererek, bakım kolaylığı ve çevrimsel karmaşıklığı hakkında istatistiksel veriler sunmuştur. Geliştirilen yazılımın son bölümünde, stil denetimi yapan modül, kodlama standartlarına göre projeleri değerlendirerek standartlara uygun olmayan kullanımları raporlamıştır.

Çalışmanın ikinci bölümünde yazılım güvenliği bilincini yaymak, güvenlik prensibini programcılara kazandırmak ve geliştiricilerin kodlar üzerinde güvenlik sınıflandırması yapmaları adına online güvenlik analiz yazılımı

geliştirilmiştir. Geliştirilen yazılım derlenmemiş kodlar üzerinde metrik analiz yaparak çevrimsel karmaşıklık, bakım endeksi ve ağırlıklı metot sayısı metriklerinin eşik değerlerine göre güvenlik derecelendirilmesini Bayes sınıflayıcı ile yapmıştır. Asp.net ile geliştirilen 5 test projesi metrikler kullanılarak sınıflandırılmıştır.

Çalışma kapsamında asp.net, php ve java da geliştirilen projeler üzerinde yapılan analiz sonucunda Xss ve Sql enjeksiyonunun en çok karşılaşılan zafiyetler olduğu tespit edilmiştir. Online güvenlik analiz yazılımında test edilen projelerin sınıflandırma başarımlarının ise %85 ile %100 arasında değişmekte olduğu görülmüştür. Yazılım güvenliği alanında aynı anda asp.net, java ve php dillerinde geliştirilen projeleri test edebilen analiz aracı ve metriklere göre güvenlik sınıflandırması yapan web uygulaması, yazılım güvenliği alanında ticari yazılımlara göre alana detaylı ve farklı bir yaklaşım sunmuştur. Ulusal yazılım güvenliği alanında statik kod analiz yazılımlarının bulunmaması ve ulusal siber güvenlik alanında yapılan çalışmaların yetersizliği göz önüne alındığında çalışmanın alana katkıda bulunacağı düşünülmektedir.

Anahtar Kelimeler: Yazılım Güvenliği, Statik Kod Analizi, Web Güvenliği, Kod Metrikleri.

2015, 141 sayfa

ABSTRACT

Ph.D. Thesis

DEVELOPMENT OF SOFTWARE SECURITY TESTING AND EVALUATION TOOL

Güncel SARIMAN

**Süleyman Demirel University
Graduate School of Applied and Natural Sciences
Department of Computer Engineering**

Supervisor: Assoc. Prof. Dr. Ecir Uğur KÜÇÜKSİLLE

With the increasing amount of internet use in our daily life, web and mobile applications have increased in numbers, too. Security and usability of these applications where users share their personal information have become to be a factor about which users should be careful. Identifying most of security vulnerabilities on the internet in developed software has increased importance of developing a secure software more and more. Detecting security vulnerabilities in applications during development can considerably decrease any impairment which may occur for software firms and users. Static and dynamic tests are used for finding out security vulnerabilities in web applications. Dynamic tests are applied to applications mostly as leakage tests. Static tests are applied by using byte or debugged code.

In this research, it is aimed to give information to programmers on code lines which include security leakage by detecting security vulnerabilities in web applications. Under the scope of this work, a testing and evaluation software was developed in order to find out the leakage in the web applications which were developed in Asp.net and Php and Java languages. The software evaluated debugged codes according to input validation, metric analysis and style checking with static code analysis method. In order to test validity of the developed software, five web projects for each ,which were presented as open source and used actively, were evaluated. With input validation, a detailed report was submitted by scanning the lines in the applications according to vulnerability types. The software scanned SQLI, XSS, CSRF, File Inclusion, Config vulnerabilities which are provided the results obtained with graphical reports and details. By giving information about resource code quality of the projects with metric analysis, the software presented statistical data about maintainability index and cyclomatic complexity. In the last section, the software, which checked style, reported uses which were not suitable by evaluating the projects according to coding standards. In the second section of this work, online security analysis software was developed in order to give information programmers about software security and help developers categorize security on codes. Bayes classifier was used for ranking security according to the thresholds of cyclomatic complexity, maintainability index and weighted method per class by applying

metric analysis on debugged codes. Five benchmark projects which were developed in Asp.net were classified by using metrics.

In this research, XSS and Sql injection were identified as mostly encountered vulnerabilities according to the results of the analysis on the projects which were developed in Asp.net, Php and Java. The success rate of classification of the projects which were tested in this online security analysis software was between %85 and %100. The analysis tool, which could test software security level in 3 languages, and the web application, which classified security according to metrics, presented a more detailed and different approach compared to commercial software. Taking into consideration the lack of static code analysis software in the field of software security in our country and very few number of researches on the field of national cyber security, it is believed that this study will contribute to the field.

Keywords: Software Security, Static Code Analysis, Web Security, Code Metrics.

2015, 141 pages

TEŞEKKÜR

“Yazılım Güvenliği Test ve Değerlendirme Aracı Geliştirilmesi” isimli doktora tez çalışmamın yürütülmesinin, sonuçlandırılmasının ve değerlendirilmesinin her aşamasında bilgi, tecrübe ve yardımlarını esirgemeyen değerli danışman hocam Doç. Dr. Ecir Uğur KÜÇÜKSİLLE’ye saygı ve teşekkürlerimi sunarım.

Bilgi güvenliği konusunda eğitim aldığım Bilgi Güvenliği Akademisi Kurumuna, güvenlik konusunda bilgilerini paylaşan Sn. Akın TOSUNLAR’a ve 3888-D1-14 No’lu Proje ile tezimi maddi olarak destekleyen Süleyman Demirel Üniversitesi Bilimsel Araştırma Projeleri Yönetim Birimi Başkanlığı’na teşekkür ederim.

Tezimin her aşamasında beni yalnız bırakmayan, maddi ve manevi desteğini benden hiçbir zaman esirgemeyen aileme, anlayış ve desteğinden dolayı eşime en içten duygularıyla teşekkür eder, sonsuz sevgi ve saygılarımı sunarım.

Güncel SARIMAN
İSPARTA, 2015

ŞEKİLLER DİZİNİ

Sayfa

Şekil 1.1. Yazılım geliştirme sürecinde yazılım açıklarının giderilme maliyeti.....	3
Şekil 1.2. Güvenli Yazılım Geliştirme Yaşam Döngüsü (G-YGYD)	4
Şekil 3.1. Web uygulama mimarisi.....	25
Şekil 3.2. Güvenlik açıklarının yıllara göre web uygulamalarında görülme ihtimali	27
Şekil 3.3. Kaynak kod analiz yazılımı blok diyagramı.....	71
Şekil 3.4. Yazılım güvenlik seviyesi belirleme yazılımı.....	73
Şekil 4.1. Kaynak kod analiz yazılımı veritabanı diyagramı.....	75
Şekil 4.2. Yazılım güvenliği test ve değerlendirme aracı girdi denetimi arayüzü	76
Şekil 4.3. Girdi denetimi arayüzü.....	102
Şekil 4.4. Girdi denetimi açıklık tipi seçim arayüzü	103
Şekil 4.5. Analiz türü seçim arayüzü.....	103
Şekil 4.6. Güvenlik seviyesi belirleme yazılımı veritabanı diyagramı.....	104
Şekil 4.7. Online güvenlik seviyesi belirleme yazılımı-1	105
Şekil 4.8. Online güvenlik seviyesi belirleme yazılımı-2	105
Şekil 4.9. Dosya yükleme ve metrik analizi.....	106
Şekil 4.10. Projede kullanılan eğitim seti	109
Şekil 4.11. Bayes sınıflandırma arayüzü	109
Şekil 4.12. Bayes sınıflandırma raporları.....	110
Şekil 4.13. Girdi analizi sonuç ekranı	115
Şekil 4.14. Girdi analizi detay penceresi	115
Şekil 4.15. Girdi analizi rapor sayfası.....	116
Şekil 4.16. Metrik analizi sonuç ekranı.....	118
Şekil 4.17. Metrik analizi rapor sonuç ekranı	118
Şekil 4.18. Stil denetimi sonuç ekranı.....	120

ÇİZELGELER DİZİNİ

Sayfa

Çizelge 3.1. Özel karakterin html denklikleri.....	22
Çizelge 3.2. Örnek kodun htmlspecialchars fonksiyonundan geçirilmesi	23
Çizelge 3.3. Örnek veri kümesi	60
Çizelge 3.4. Eğitim kümesi.....	60
Çizelge 3.5. Olasılık hesaplamaları.....	61
Çizelge 3.6. Hazır Düzenli İfade Desenleri	61
Çizelge 3.7. Karakter katarları	62
Çizelge 4.1. Sql enjeksiyonu tespit etme çalışma altyapısı	77
Çizelge 4.2. Xss saldırısı tespit etme sistemi çalışma altyapısı	78
Çizelge 4.3. İstek doğrulama tespit etme sistemi çalışma altyapısı	79
Çizelge 4.4. Toctou zafiyeti tespit etme sistemi çalışma altyapısı	80
Çizelge 4.5. Xpath enjeksiyonu tespit etme sistemi çalışma altyapısı.....	81
Çizelge 4.6. Ldap enjeksiyonu tespit etme sistemi çalışma altyapısı.....	81
Çizelge 4.7. Log hatası tespit etme sistemi çalışma altyapısı.....	84
Çizelge 4.8. Serileştirme problemi tespit etme sistemi çalışma altyapısı.....	85
Çizelge 4.9. Php dilinde xss saldırısı tespit etme sistemi çalışma altyapısı	86
Çizelge 4.10. Php dilinde xpath enjeksiyonu tespit etme sistemi çalışma altyapısı	87
Çizelge 4.11. Php dilinde log hatası tespit etme sistemi çalışma altyapısı	88
Çizelge 4.12. Php dilinde dosya sızıntısı tespit etme sistemi çalışma altyapısı...	89
Çizelge 4.13. Java dilinde xss saldırısı tespit etme sistemi çalışma altyapısı	92
Çizelge 4.14. Java dilinde race condition saldırısı tespit etme sistemi çalışma altyapısı	94
Çizelge 4.15. Çevrimsel karmaşıklık eşik değerleri.....	107
Çizelge 4.16. NOM eşik değerleri	107
Çizelge 4.17. Bakım endeksi eşik değerleri.....	107
Çizelge 4.18. Eşik değer aralıklarına göre metrik puanları	108
Çizelge 4.19. Asp.net projelerinin girdi analizi sonucunda elde edilen test sonuçları	111
Çizelge 4.20. Php projelerinin girdi analizi sonucunda elde edilen test sonuçları	112
Çizelge 4.21. Java projelerinin girdi analizi sonucunda elde edilen test sonuçları	114
Çizelge 4.22. asp.net, php ve java projelerinin metrik analizi sonucunda elde edilen test sonuçları	116
Çizelge 4.23. asp.net, php ve java projelerinin stil denetimi sonucunda elde edilen test sonuçları	119
Çizelge 4.24. Online güvenlik yazılımı test sonuçları	120
Çizelge 5.1. Asp.net test projelerinde zafiyet barındıran toplam satır sayısı.....	124

Çizelge 5.2. Php test projelerinde zafiyet barındıran toplam satır sayısı	124
Çizelge 5.3. Java test projelerinde zafiyet barındıran toplam satır sayısı	125
Çizelge 5.4. Asp.net test projelerinde zafiyet barındıran toplam satır sayısı.....	125
Çizelge 5.5. Java test projelerinde zafiyet barındıran toplam satır sayısı	126
Çizelge 5.6. Php test projelerinde zafiyet barındıran toplam satır sayıları	126



SİMGELER VE KISALTMALAR DİZİNİ

BLOC	Boş Satır Sayısı (Blank Line of Code)
CBO	Nesne Sınıfları Arasındaki Bağımlılık (Coupling Between Objects)
CC	Çevrimsel Karmaşıklık (Cyclometric Complexity)
CK	Chidamber ve Kemerer
CLOC	Yorum Satırı Sayısı (Comment Line of Code)
CP	Yorum Oranı (Comment Percentage)
CSRF	Siteler Arası İstek Sahteciliği (Cross-Site Request Forgery)
CSS	Basamaklanmış Stil Katmanları (Cascading Style Sheet)
DDOS	Dağıtık hizmet engelleme (Distributed Denial of Service attack)
DIT	Kalıtım Ağacının Derinliği (Depth of Inheritance)
DNS	Alan Adı Sistemi (Domain Name Server)
DOS	Hizmet Engelleme (Denial of Service attack)
G-YGYD	Güvenli Yazılım Geliştirme Yaşam Döngüsü
HTML	Hiper Metin İşaretleme Dili (Hypertext Markup Language)
HV	Halstead Sayısı (Halstead Volume)
IDP	Saldırı Tespit Sistemi (Intrusion Detection and Prevention)
LCOM	Metotların Uyumluluğu (Lack of Cohesion)
LDAP	Dizin Servisi (Lightweight Directory Access Protocol)
LFI	Yerel Dosya Çağırma (Local File Inclusion)
LOC	Kod Büyüklüğü (Line of Code)
MINDEX	Bakım İndeksi (Maintainability Index)
NOC	Alt Sınıf Sayısı (Number of Children)
NOM	Sınıf Başına Method Sayısı (Number of Method)
OSI	Açık Sistem Ara Bağlantı Modeli (Open Systems Interconnection)
OWASP	Açık Web Uygulama Güvenliği Projesi (Open Web Application Security Project)
RDBMS	İlişkisel Veritabanı Yönetim Sistemi (Relational Database Management System)
RFC	Sınıfın Tetiklediği Metot Sayısı (Response for a Class)
RFI	Uzak Dosya Çağırma (Remote File Inclusion)
SLOC	Kaynak Kod Büyüklüğü (Source Line of Code)
SOME	Siber Olaylara Müdahale Ekipleri
SQLI	Yapısal Sorgulama Dili Enjeksiyonu (SQL Injection)
SSL	Güvenli Giriş Katmanı (Secure Socket Layer)
SVI	Güvenlik Endeksi (Security Index)
SYN	Hizmet Engelleme Saldırı Tipi
USOM	Ulusal Siber Olaylara Müdahale Merkezi
WMC	Sınıf Başına Ağırlıklı Metot Sayısı (Weight Method of Class)
XPATH	XML Sorgulama Dili (XML Path Language)
XSS	Çapraz Site Betikleme (Cross Site Scripting)
YGYD	Yazılım Geliştirme Yaşam Döngüsü

1. GİRİŞ

Son yıllarda yaşanan teknolojik gelişmeler ile birlikte bilgisayarlar hayatımızın her alanına girmiş ve “olmazsa olmaz” bir şekilde kullanılmaya başlanmıştır. İnternetin yaygınlaşmasıyla web ve mobil uygulamaların çoğalması, günlük hayatı bankacılık işlemleri, savunma sistemleri, rezervasyon işlemleri gibi birçok alanda oldukça kolaylaştırmaktadır. Teknolojik gelişmeler ve uygulamaların farklı kullanım alanları, sistemleri aynı zamanda saldırganların açık hedefi haline getirmektedir. Önemli ve riskli işlemlerin yapılabildiği bilgisayar sistemlerinin korunması da en az teknolojinin sağladığı avantajlar kadar dikkate alınması gereken bir konudur.

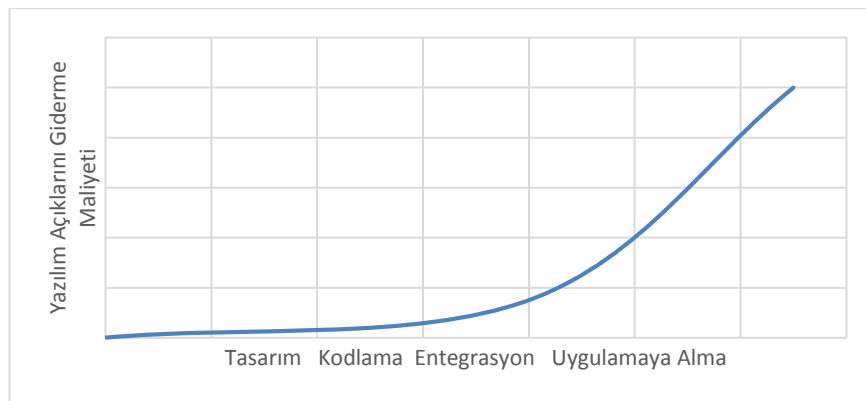
Bilgi güvenliği, bilginin izinsiz veya yetkisiz bir biçimde erişim, kullanım, değiştirilme, ifşa edilme, ortadan kaldırılma, el değiştirme ve hasar verilmesini önlemek olarak tanımlanırken gizlilik, bütünlük ve erişilebilirlik olarak isimlendirilen üç temel güvenlik ögesinden herhangi birinin zarar görmesi güvenlik zafiyeti oluşturmaktadır (Pesen, 2013). Bilginin sadece izin verilen kişilerin izin verilen yollarla erişimi bilginin gizliliğiyle sağlanmaktadır. Bilginin, her şeyden önce doğru ve kesin oluşu, şüphe uyandırmayan bir durumda olması bütünlüğü sağlarken aynı zamanda, bilginin anlamlı ve tutarlı olmasını da gerektirmektedir. Erişilebilirlik ise bilginin yetkili kişilerce ihtiyaç duyulduğunda ulaşılabilir ve kullanılabilir durumda olmasıdır. Kritik altyapıları korumak için bilgi güvenliği hem kamu sektörü hem de özel sektör için çok önemlidir. Bilgi ve bilgisayar güvenliğinde, karşı taraf, kötü niyetli olarak nitelendirilen kişiler ve yaptıkları saldırılardır. Var olan bilgi ve bilgisayar güvenliği sistemini aşmak veya atlatmak, zafiyete uğratmak, kişileri doğrudan veya dolaylı olarak zarara uğratmak, sistemlere zarar vermek, sistemlerin işleyişini aksattırmak, durdurmak, çökertmek veya yıkmak gibi kötü amaçlarla yapılan girişimler saldırı veya atak olarak adlandırılmaktadır. Saldırganlar, amaçlarına ulaşmak için çok farklı teknikler içeren saldırılar gerçekleştirmektedirler. Saldırı türlerinin bilinmesi, doğru bir şekilde analiz edilmesi ve gereken önlemlerin belirlenmesi, bilgi güvenliği için büyük bir önem arz etmektedir (Çebi, 2013).

Bilgi güvenliği Securosis araştırma firmasının yaptığı sınıflandırmaya göre ağ, güvenliği, uç nokta güvenliği, veri güvenliği, uygulama güvenliği, kimlik ve erişim güvenliği, güvenlik yönetimi ve sanallaştırma kategorilerine ayrılmıştır (Pesen, 2013). Uygulama güvenliği kategorisinde yazılımların geliştirilmesi ve uygulanması sırasında meydana gelebilecek açıklıklar incelenmektedir. Web uygulama güvenlik duvarları, uygulama testi, güvenli geliştirme, tehdit modelleme, geliştirme süreci, test metodolojileri, web sızma testleri, web uygulama güvenlik duvarları, web uygulama değerlendirme ve yönetilir servisler uygulama güvenliği kategorisinde alt dallara ayrılmıştır.

Tüm dünyada hızla gelişen teknolojik gelişmelerle birlikte ülkemizde de bu gelişmeler hayatımızın her alanında önemli roller oynamaktadır. Bilgi sistemlerinde bulunan güvenlik zafiyetleri, sistemlerin hizmet dışı kalmasına veya kötüye kullanılmasına, can kaybına, büyük ölçekli ekonomik zarara, kamu düzeninin bozulmasına sonucunda da ulusal güvenliğin ihlaline neden olabilmektedir. Bu bilgiler ışığında Bakanlar kurulunca ulusal siber güvenlik çalışmalarının yürütülmesi, yönetilmesi ve koordinasyonuna ilişkin karar alınmıştır. 7/24 müdahale esasına göre çalışacak Ulusal Siber Olaylara Müdahale Merkezi (USOM) kurularak, USOM' un koordinasyonunda çalışacak sektörel Siber Olaylara Müdahale Ekipleri (SOME) oluşturulması alınan kararlar içerisinde yer aldığı toplam 29 alt başlıkta hazırlanmıştır. Eylem planı kapsamında yazılım güvenliği ile ilgili eğitimlerin hazırlanması, kritik alt yapılarda kullanılmak üzere geliştirilen yazılımlar için programlama dili bağımsız güvenli yazılım geliştirme kuralları dokümanının hazırlanması planlanmıştır.

Bilgi sistemlerindeki güvenlik zafiyetleri geliştirilen uygulamalar üzerinde yaygınlaşmaktadır. 2005 yılından itibaren kayda geçirilen bilişim güvenliği ihlalleri olaylarına göre, Privacy Rights Clearinghouse adlı dernek tarafından her yıl büyük bir bilgi hırsızlığı olayı yaşandığı tespit edilmiştir. Dernek tarafından 226 milyon bilişim güvenliği ihlali raporlanmıştır. Gartner adlı teknoloji araştırma ve danışmanlık firması bilişim güvenliği ihlalleri ile alakalı yaptığı araştırmada, bu ihlallerin %80 oranında yazılım güvenliği problemlerinden

kaynaklandığını ortaya koymuştur (Karayumak, 2013). Yazılımı oluşturan birçok bileşen ve dinamik yapı olduğu için, tam güvenli yazılım diye bir kavram geçerli bir açıklama değildir. Fakat, yazılımlarda güvenlik zafiyetlerinin temel kaynakları vardır. Yazılımdaki zafiyetlerin temelinde kodlama ve mimari hatalar bulunmaktadır. Yazılım Güvenliği, yazılımların tersine mühendislik yöntemleri ve araçları ile algoritmaların ortaya çıkarılması veya değiştirilmesini engellemeyi amaçlayan yöntemler bütünüdür. Uygulama geliştiricilerin yazılım güvenliği bilgilerine sahip olmamalarından dolayı, yazılım güvenliğinin en zayıf halkasını programlar oluşturmaktadırlar. Uygulamaların diğer bileşenleriyle ve işletim sistemleriyle bütünleşme noktaları olan programlar saldırganlar tarafından sıkça kullanılmaktadır. Yazılımların yaygın olarak kullanılmaya başlandığı ilk yıllarda kaliteli ve olgun yazılım üretmek, son yıllarda ise özellikle güvenli yazılım geliştirmek için çok sayıda model ve çerçeve üzerinde çalışılmıştır. Bu durumun en büyük tetikleyicisi son yıllarda güvenlik açıklıklarının artmasıdır (Alparslan, 2009). Yazılım geliştirmede erken bir süreçte farkına varılan yazılım açıklıklarının düzeltilmesinin daha ileri süreçlerde farkına varılan açıklıklara göre daha az maliyetli olacağı yazılım endüstrisinde yaygın olarak kabul edilen bir ilkedir (Michael, 2005). Bu ilke yazılım geliştirme sürecinin güvenli olmasının maliyet açısından da nedenli önemli olduğunun göstergesidir. Şekil 1.1' de yazılım açıklıklarının süreç boyunca giderilme maliyetlerinin grafiği verilmiştir.



Şekil 1.1. Yazılım geliştirme sürecinde yazılım açıklıklarının giderilme maliyeti

Geleneksel olarak, yazılım güvenliği, yazılım geliştirme yaşam döngüsünün (YGYD) başlarında dikkate alınmaz ve sadece geliştirmenin sonraki

aşamalarında yazılıma entegre edilmeye çalışılması nedeniyle, yazılım geliştirmenin çeşitli aşamalarında güvenlik açıklarının risklerini arttırmaktadır (McGraw, 2006). Güvenli yazılım mühendisliği, YGYD' nin başında ve ilerleyen aşamalarında güvenlik problemlerini dikkate alarak güvenlik açıklarını önlemeyi amaçlamaktadır. Güvenli yazılım mühendisliği tasarım, yapı ve test süreciyle güvenilirliği sağlamaktadır. Bir Güvenli Yazılım Geliştirme Yaşam Döngüsü (G-YGYD) süreci yazılım geliştirme yaşam döngüsü boyunca güvenli yazılım geliştirme yöntemlerini kullanan güvenlik yöntemlerini dikkate alır (Sarıman ve Küçüksille, 2013). Genel olarak problemlerin çoğu, yazılım geliştirme sürecinde gereksinim ve sistem analizlerinin doğru ve yeterli yapılmamasından kaynaklanmaktadır. Yazılımlardaki analiz eksikliği güvenlik riski oluşturmakta, bu durum bilgiye yönelik tehditlerin ortaya çıkmasında önemli bir açıklık oluşturmaktadır. Yazılım geliştirme yaşam döngüsünün temel ayağı olan 4 genel yaklaşım bulunmaktadır. YGYD metodolojisi şelale modeli, artırılmış model, spiral model, prototip modeli olarak sınıflandırılmaktadır. Güvenlik prensibini YGYD' ne dâhil etmek için yaşam döngüsünün her aşamasında güvenliği ele almak gerekmektedir. G-YGYD modeli şekil 1.2' de gösterilmiştir.



Şekil 1.2. Güvenli Yazılım Geliştirme Yaşam Döngüsü (G-YGYD)

Yazılım geliştirme sürecinde, yazılımın sorunsuz bir şekilde çalışabilmesi için test aşaması tüm süreç boyunca değerlendirmeye alınmalıdır. Geliştirilen bir bilgisayar yazılımının kalitesi, güvenliği ve doğruluğu test süreciyle kontrol edilebilmektedir. Yazılım testi, yazılımdaki hataları bulmak için bir programın ya da uygulamanın çalıştırılma süreci olmakla birlikte eleştirme veya karşılaştırma işlemidir (Widera, 2006). Yazılım testi, ileriye dönük kodun geliştirilme masraflarını azaltmak, ürün çalıştırılmadan önce kalitesini ve senaryolara uygunluğunu denetlemek, geliştirme sırasında gözden kaçan yanlışları bularak bu yanlışların ileride de tekrarlanmasını önleyerek zaman ve maliyet tasarrufu yapmaktır. Yazılım testleri ile hatadan arındırılmış, daha kaliteli ve güvenilir yazılımlar üretilmesi hedeflenmektedir. Yazılım testlerinde üç temel test tekniği

vardır ve genel olarak kutu yaklaşımı olarak sınıflandırılmaktadır. Kutu yaklaşımı, yazılımın girdi ve çıktıları veya algoritma ve değişkenlerinin test edilmesine göre belirlenmiştir. Kutu yaklaşımında kara kutu, beyaz kutu ve gri kutu test teknikleri kullanılmaktadır.

Kara kutu testinde, yazılım derlenmiş kod üzerinden test edilirken kod ve tasarım hakkında bilgi sahibi olmaya ihtiyaç duyulmamaktadır. Test edilecek olan uygulamanın kodu hiç dikkate alınmadan, sadece girdilerin ve çıktıların incelenmesi ile gerçekleştirilen test metodudur. Bu yaklaşımda, test edilecek olan yazılımın, sonucu bilinen bir davranışını doğrulamak için, davranışın gerektirdiği girdi değerleriyle çalıştırılarak sınanır. Daha sonra yazılımın bu girdi karşısında elde edilen çıktısı beklenen sonuçla karşılaştırılır. Testçinin testi uygularken yazılımın içyapısı ile kesinlikle ilgilenmemesinden dolayı yaklaşımın adı kara kutu testi olarak belirlenmiştir.

Beyaz kutu testi ile kod içerisinde gizli kalmış mantıksal hatalar tespit edilerek, yazılan kodun optimizasyonuna katkıda bulunulur. Kod içindeki fazla satırlar ayıklanarak ölü kod parçaları bulunur. Statik testlerle yazılan kodların elle veya yardımcı yazılımlarla hatalara karşı analiz edilmesi sağlanır. Beyaz Kutu testleri yazılım kod ifadeleri, köprüleri, yolları, durumları gibi konuları kapsamaktadır. Kaynak kod beyaz kutu testlerinin en önemli girdisi olduğundan koda ulaşım olmadan beyaz kutu testleri gerçekleştirilemez. Statik kod analizleri yazılımın kalitesi, çalışma performansı, işlevselliği, güvenilirliği açılarından önemlidir. Dinamik testler kapsamında sistemin değişen veriler karşısında nasıl tepki verdiği gözlemlenir. Dinamik testlerin gerçekleştirilmesi için yazılan kod derlenir ve koşturulur. Dinamik testler performans testleri, uyumluluk testleri, zorlama (stres) testleri, uygunluk testleri, duman testleri, sürüm doğrulama testleri başlıklarında çalışmaktadır.

Gri kutu testleri ise beyaz kutu tekniği ile kara kutu test tekniklerinin birlikte kullanılmasıdır. Kara kutu testleri sistemin içyapısı bilinmeden gereksinimler temel alınarak gerçekleştirilir; beyaz kutu testinde ise sistemin içyapısı tam olarak bilinir ve testler bu içyapı üzerinden gerçekleştirilir. Gri kutu testinde ise, gereksinimleri doğrulayacak test durumları kodun içyapısı esas alınarak yazılır.

Böylece gereksinimlerin gerçekleştiği doğrulandığı gibi yazılımın içyapısı da sınanmış olur.

İnternet teknolojisi günümüzde güncel ve doğru bilgiyi insanlara ulaştırmak için en kolay ve en etkin yöntem olarak bilinmektedir. Web uygulamalarının yaygınlaşması, bu uygulamaları geliştirmek için yeni teknolojilerin çıkması ve web uygulamalarının son kullanıcıya sağladığı hizmetlerin daha geniş bir perspektife yayılması, uygulamaların güvenli kılınması noktasını daha da belirgin hale getirmektedir. E-Devlet uygulamaları, internet bankacılığı işlemleri, çevrimiçi alışveriş ve benzeri birçok web uygulamasının ise hayatımızda önemli rol oynamaya başlaması, hatta sağlık ve finans sektörleri gibi doğrudan kişileri etkileyen sektörlerin web uygulamaları aracılığıyla hizmet vermesi, hem kullanıcı bazlı kişisel güvenliğin önemini artırmakta hem de bu hizmetin kullandığı web uygulamalarının güvenli olmasını zorunlu hale getirmektedir (Yılmaz, 2013). Web sitesi güvenliğinin yeterince ciddiye alınmaması, site saldırılarını her geçen gün arttırmaktadır. Web sistemlerin birçoğunda çapraz betik sorgulama (XSS) ve yapısal sorgulama dili enjeksiyonu (SQLi) başta olmak üzere birçok güvenlik açığı bulunmaktadır. Kullanılan PhpBB veya Joomla gibi hazır web sitesi geliştirme ortamları birçok güvenlik açığını barındırmakta ve yanlış ayarlamalar sonucunda sistemler kolaylıkla kırılabilir (Ulakbim, 2009).

Geliştiricilerin web uygulamalarına ne tür saldırılar gelebileceği konusunda bilgisiz olması, saldırının nerelerden gerçekleşebileceğini görememesi gibi nedenlerden ötürü uygulamalar saldırılara açık hale gelmektedir. Bu sürece katkıda bulunan kişilerde "güvenlik farkındalığının oluşması", "saldırıları konusunda bilgi edinilmesi" ve "uygulamanın bu verilere göre revize edilip güvenli hale getirilmesi" sorunları büyük ölçüde çözmektedir. Web açıklıkları genel olarak; hizmet engelleme (Dos) atakları, dağıtık hizmet engelleme(D-Dos), virüsler, sistemsel açıklar (işletim sistemi, uygulama sunucusu, güvenlik duvarı, veritabanı), yazılım geliştirici nedenli açıklardan oluşmaktadır. Geliştiriciler tarafından yapılan yanlışlar ise, girdi kontrollerinden oluşmaktadır.

Kurumların hizmet sunumlarında bilgi sistemlerini her geçen gün daha fazla kullanmaları ile birlikte, söz konusu sistemlerin güvenliğinin sağlanması siber

güvenliği önemli bir seviyeye taşımıştır. Bilgi sistemlerin en önemli parçası olan yazımların güvenliği alanında akademik çalışmaların yetersizliği ve saldırı tiplerinin gün geçtikçe değişmesi, yazılım güvenliğinin ulusal siber güvenlik stratejilerinin önemli bir parçası olması, bu alanda yapılacak çalışmaları önemli kılmaktadır.

Bu tez çalışmasının amacı, güvenlik sürecinin, yazılım geliştirme sürecine entegre edilmesidir. Yazılımlarda, sonradan ortaya çıkan açıklara yama geliştirmek, bu açıkların daha yazılım geliştirme sürecinde fark edilip önlemeye nazaran çok daha fazla mali yük getirmesi G-YGYD' nin mali açıdan da bir gereksinim olduğunu ortaya koymaktadır. Web uygulamalarının yaygınlaşması ve hayatımızda önemli rol oynamaya başlamasıyla web uygulamalarının da güvenli olması zorunlu hale gelmiştir. Bu çalışmada web projelerinde güvenli yazılım geliştirmek için dikkat edilmesi gereken süreçlerin ortaya çıkarılması amaçlanmıştır. Bir web uygulamasının güvenli bir şekilde tasarımı ve geliştirilmesi için XSS, url güvenliği, oturum mekanizması ve kod güvenliği gibi açıkların raporlanabileceği ve bu açıkların düzeltilebileceği bir uygulama geliştirilmiştir. Yazılım metrikleri kullanılarak yeni bir güvenlik derecelendirme sisteminin geliştirilmesi de bu çalışma kapsamında ele alınmıştır.

Çalışma kapsamında geliştirilen test ve değerlendirme aracının java, asp, asp.net ve php dillerinde geliştirilen web tabanlı yazılımların statik kod analizini yaparak girdi denetimlerini, kod metriklerini ve stil denetimlerini gerçekleştirmektedir. Geliştirilen test aracı, sisteme yüklenen derlenmemiş proje kaynak kodları incelerken düzenli ifadelerden faydalanılmıştır. Test ve Değerlendirme aracı, yazılım geliştiricilerin kullanabileceği bir sistem olduğu için güvenlik göz önüne alınarak masaüstü uygulaması şeklinde geliştirilmiştir. Bu sayede kullanıcılar projeyi kendilerine ait bilgisayara kurarak local bir test aracı kullanmış olacaklardır. Test aracı geçmişe dönük raporlamalar sunarak geliştiricinin kodlama bilinci de raporlardan ortaya çıkarılabilmektedir. Çalışmanın diğer bölümünde ise tüm geliştiricilere faydalı olması ve yazılım güvenlik eğitimlerinde kullanılabilmesi amacıyla web tabanlı metrik analiz yazılımı geliştirilmiştir.

Yazılımların kod kalitesinin ölçülebildiği sistem aynı zamanda naive bayes algoritmasını kullanarak güvenlik derecelendirmesi yapabilmektedir.

Test ve değerlendirme aracından alınan sonuçlar, piyasada var olan analiz araçlarından daha farklı ve güncel sonuçlar ortaya koymaktadır. Yazılım dillerinde her geçen gün artan uygulama çatıları piyasadaki birçok analiz aracına uyumlu değildir. Ayrıca ulusal bir yazılım test ve değerlendirme yazılımının olmaması geliştirilen projeyi ulusal alanda önemli kılmaktadır.



2. KAYNAK ÖZETLERİ

Yapılan literatür araştırmalarıyla kaynaklar bilgi güvenliği, yazılım güvenliği, statik kod analizi, web uygulama güvenliği ve yazılım güvenlik testleri olmak üzere beş ana başlıkta toplanmıştır. Makaleler, e-kitaplar, yüksek lisans ve doktora tez çalışmaları araştırmaya yol göstermektedir.

Myers (2004) “The Art of Software Testing” kitabında, yazılan programların testleriyle ilgili pratik bilgiler vermektedir. Ayrıca proje yöneticileri ile ilgili test sürecinde dikkat edilmesi gereken noktalara değinmektedir. Öğrenciler için programların testlerinde maruz kalabilecekleri durumları incelemektedir. Kitabın, programlama kurslarında öğrencilerin yazılım testleri aşamasında karşı karşıya kalabileceği problemleri, kursun erken aşamalarında aşmaları adına yardımcı bir doküman olabileceği görülmüştür.

Livshits (2005), çalışmasında java bileşenlerinde sıkça meydana gelen güvenlik açıklarına değinmiştir. Servletlerde ve Tomcat sunucularda meydana gelen kötü oturum kayıtları ve SQLi zafiyetinin statik kod analizi ile giderilebileceği gösterilmiştir. Eclipse kod derleyicisine ait statik analiz eklentisi kullanılarak 10 açık kaynak kodlu java uygulaması analiz edilmiştir. Analiz sonucunda %37 yanlış pozitifle birlikte 14 kötü oturum kayıt hatası bulunmuştur. Eclipse, sql injection saldırısını 3 farklı kategoride incelemektedir. Benchmarklar üzerinde bu kategoriler incelenmiş ve hata sayısı 188 olarak bulunmuştur. Oturum kayıtlarının hataları için bulunan yanlış pozitiflerin işaretçi analizi kullanılarak azaltılabileceği tespit edilmiştir. Sql enjeksiyonlarının elle tespit edilmesinin zor olduğu, bulunan birkaç hatanın yanlış olduğu ve gözden kaçtığı görülmüştür. Bunun yerine yazılım araçlarının kullanılmasının daha doğru sonuç vereceği ön görülmüştür.

Gallagher vd. (2006), “Hunting Security Bugs” kitabında saldırgan olarak davranıldığında yazılımlardaki potansiyel güvenlik açıklarının neler olabileceğini ortaya çıkarmaya çalışmıştır. Kitap beyaz şapkalı saldırganlar için yazılırken, zafiyetlerin de nelerden kaynaklandığını ve nasıl ortaya çıkarılabileceğini

anlatmaktadır. Kitapta; yüksek risk barındıran giriş noktalarının tanımlanması ve test durumlarının oluşturulması, güvenlik açıklarını ortaya çıkarmak için siyah kutu ve beyaz kutu yaklaşımlarının kullanılması, programın mantığı ile uyuşan sql enjeksiyonu gibi açıkların tespit edilmesi ve saldırganların hafızaya nerelerde müdahale edebilecekleri belirtilmiştir. Kitapta bahsedilenlerin aksine, yazılımların nasıl daha güvenli hale getirilmesi gerektiği bu kapsamda anlatılmamaktadır.

Livshits (2006), doktora tez çalışmasında yazılım güvenliğini statik ve çalışma zamanı analizleriyle geliştirmeye çalışmıştır. Web uygulamalarının gelişmesiyle ara bellek taşmaları oldukça yaygınlaşmıştır. Tez çalışması, web güvenlik açıkları geniş bir çerçevede statik ve run time bir çözüm olarak tanımlanmıştır. Çalışma web tabanlı java uygulamalarını incelemek için geliştirilmiştir. Geliştirilen statik kod denetim aracı derleyicide yazılan java byte kodlarını denetleyerek açıkları raporlayabilmektedir. Çalışmanın sonunda 11 test uygulamasından yanlış positiflerle birlikte 98 güvenlik hatası bulunmuş ve çalışma sonucunda elde edilen sonuçlar tartışılmıştır.

Jovanovic vd. (2006), çalışmalarında son yıllardaki web uygulamalarının zayıflık tespitine statik kod analizi ile yaklaşmışlardır. Yazılımdaki zayıflıkları bulabilmek için flow-sensitive, interprocedural ve veri akış analizi yöntemleri kullanılmıştır. Geliştirilen uygulama php dilinde yazılan kodlara ait SQLI ve XSS zafiyetlerini tespit etmektedir. Çalışma sonunda 3 farklı web uygulamasında daha önce tespit edilemeyen 15 zafiyet ve 36 adet bilinen zafiyet bulunmuştur. Yanlış pozitif oranı ise % 50dir.

Urgun (2007) yazmış olduğu uygulama kılavuzunda, web uygulama güvenliğini ve uyulması gereken güvenlik esaslarını anlatmıştır. Bu kapsamda güvenli kodlama prensiplerinden, güvenlik ilkelerinden, tehdit risk modellemeyen bahsetmiştir. Ayrıca web servis güvenliği, erişim kontrolleri, gizlilik, etki alanı birleşimi gibi konularda detaylı bilgiler vermiştir. Kılavuzda web uygulamasının güvenli bir şekilde tasarımı ve geliştirilmesi için gerekli esaslar belirtilmektedir.

Costa vd. (2007), Bouncer yazılımını kullanarak atakları tespit etmeyi ve hedef zayıflıkları kullanarak sızmaları engellemek için otomatik filtre uygulamışlardır. Önceki çalışmalarda bilinen zafiyetlere göre filtre uyguladığı için saldırganlar rahatlıkla bu filtreyi aşabiliyordu. Bouncer yazılımı bu aşmayı engellemek için statik ve dinamik analiz, girdileri yorumlayabilen kütüphane fonksiyonları ve sembolik teknikler olmak üzere 3 farklı teknik kullanmaktadır. Sonuç olarak uygulanan filtreler, yazılımlar saldırı altında olsa bile rahatlıkla kullanımını sağlayabileceklerdir.

Tiftik vd. (2007)'nin sunmuş oldukları bildiri sistem ve yazılım geliştirme sürecinde önemli bir yere sahip olan doğrulama çalışmalarının neler olduğunu, anahtar alanlarını, uygulama aşamasında dikkat edilmesi gereken noktaları ve ASELSAN Mikrodalga ve Sistem Teknolojileri grubu'ndaki uygulama örneklerini içermektedir. Bildiride doğrulama yönetimi ile ilgili kilit noktalara vurgu yapılarak, testlerin hazırlanmasından gerçekleştirilmesine kadar olan süreç pratik deneyimler ışığında detaylandırılmaktadır. Bunun yanı sıra büyük boyutlu projelerde hatasız yazılımlara erişebilmenin öneminden bahsedilerek, bu amaca ulaşmak için hangi yöntemler ve araçların kullanılabileceği veya geliştirilebileceği ile ilgili öneriler verilmektedir.

Erdogan (2009), web uygulamalarındaki güvenlik açıklarıyla ilgili bir tez çalışması gerçekleştirmiştir. Web uygulamalarının güvenlik testi için Agile Güvenlik Testi yöntemi seçilmiş ve G-YGYD döngüsüne dahil edilmiştir. Zafiyet içeren 9 sınıf, önerilen metodolojiyle çeşitli test araçları kullanılarak test edilmiştir. Analiz sırasında testin ne kadar zaman aldığı, bulunan zafiyet sayısı ve azaltılan yanlış pozitif sayısı analiz edilmiştir. Önerilen yaklaşımdaki zaman kazancı %21 daha fazla iken, zafiyetleri bulma konusunda %95 daha etkili olduğu tespit edilmiştir. Önerilen yöntemin normalde göz ardı edilen yukarıdaki önemli faktörleri azalttığı da çalışma kapsamında elde edilen sonuçlar arasındadır.

Huming vd. (2011) yaptıkları çalışmada bilgisayar mühendisliği birinci ve ikinci sınıf öğrencilerine güvensiz kod geliştirmenin zararlarının yanında, güvenli programlama hakkında önemli bilgiler kazandırmayı hedeflemişlerdir. Yapılan

anket ve geri bildirimler, öğrencilerin meslek hayatlarında bu çalışmanın büyük önemi olduğunu göstermektedir. Çalışmanın ilk bölümünde bilgi güvenliğinden ve güvenli kod geliştirmenin öneminden bahsedilmiştir. Kurs programı kapsamında öğrenciler güvenli yazılım gerekliliğini anlayarak güvenli yazılım geliştirmeye ilgili önemli bilgiler kazanmışlardır, Kurs programında sayı taşması, girdi hataları ve bunların çözümleriyle ilgili eğitim verildikten sonra uygulama bölümünde ise verilen hatalı kodların güvenli koda dönüştürülmesi için bir laboratuvar çalışması yapılmıştır. Çalışma sonunda yapılan anket çalışmasında ön anket çalışmasına göre her bir soru için modüllerde verilen konulara yaklaşım ve kazanımın 1.5 kat arttığı gözlemlenmektedir. Bu çalışma güvensiz kod ve güvenli programlama alıştırmalarının öğrenciler üzerinde olumlu etki yarattığını göstermiştir.

Tatlı (2011), makalesinde güvenlik sürecinin, yazılım geliştirme sürecine entegre edilmesinden, entegrasyon işlemi sonucunda G-YGYD süreçlerinin ortaya çıkmakta olduğundan bahsetmektedir. GYG süreçleri sayesinde, tehdit modelleme, risk analizi, yazılımcılar için güvenlik eğitimleri ve dokümanları, mimari analiz, kod analizi, sızma testleri gibi güvenlik için vazgeçilmez gereksinimlerin bir bütün olarak yazılım sürecine entegre edildiği belirtilmiştir. Bu sebeple, bir yazılım projesine başlamadan önce projenin yapısına ve eldeki kaynaklara uygun bir GYG sürecinin belirlenmesi gerektiği ve bu sürecin gerektirdiği güvenlik aktivitelerinin uygulamaya geçirilmesi gerektiği belirtilmiştir.

Makalede ayrıca, güvenli yazılım geliştirme metotlarından ve Java'nın mimari tasarımında güvenliği desteklemek adına birçok önlemler (tip koruması, otomatik bellek yönetimi, dizi sınır kontrolü, byte code denetimi, imzalı appletler gibi) dikkate alındığı belirtilmiş ancak yazılımcıların da geliştirme süreci esnasında dikkat etmeleri gereken temel noktaların olduğu ifade edilmiştir. Bu noktaların başlıca Tasarım, Gizlilik ve Mahremiyet, Erişim Kontrolü, Girdi Denetimi ve Serileştirme başlıkları altında topladığından bahsetmektedir. Java yazılımcıları için güvenlik farkındalığı yaratmak adına hazırlanan dokümanın faydalı olacağı düşünülmüştür.

Zainuddin ve Aziz (2011) çalışmalarında yazılım güvenlik açıklarına dikkat çekmişlerdir. Son 10 yılda artan güvenlik farkındalığı, mühendislerin güvenli yazılım geliştirebilmeleri için beceri ve yeteneklerini arttırabilecek bir araç geliştirilmesini zorunlu kılmıştır. Geliştirilen araç yazılım açıklıklarını alıştırmalarla, soru, sorgu ve çözüm önerileriyle kullanıcılara daha anlaşılır bir yazılım güvenlik farkındalığı sunmaktadır. Geliştirme aşamasında öncelikle en çok bulunan yazılım açıkları ve kodlama hataları tespit edilmiştir. Yazılım, Açık Web Uygulama Güvenliği (OWASP) kuruluşunun en çok tespit ettiği hafıza taşması, oturum yönetimi ve XSS gibi açıkları listeler. Öğrenme aracı olarak tasarlanan araç, benzerlerinden kolay kullanıcı ara yüzü ile ayrılmaktadır. Ayrıca kodlamaya yeni başlayan yazılım geliştiriciler için tasarlanmıştır. Proje, kullanıcı hatalarını yazılım derlenmeden ortaya çıkaracağı için piyasadaki benzer araçlara göre kullanıcılara maliyeti daha az analiz ve çözüm ortaya koymaktadır.

AL-Amro ve El-Qawasmeh (2012) çalışmalarında, Asp.net ile geliştirilen web sitelerinin güvenlik açıkları olabileceğinden ve bu durumun geliştirici tarafından dikkate alınmayacağından bahsetmektedir. Çalışmada, geliştirilecek olan algoritmanın güvenlik açıklarını tespit edebilmesi hedeflenmiştir. Geliştirilen tarama aracı ile Asp.net web sayfalarında en sık rastlanan güvenlik açıklarının tespit edilmesi hedeflenmiştir. SQLi ve XSS zafiyetlerini bulmak için 12 aşamadan oluşan bir algoritma çıkarılmıştır. Geliştirilen uygulamayı test etmek için internetten indirilen kaynak kodlar kullanılmıştır. Çalışmanın sonucunda yapılan analizle ilgili rapor çıkartılırken, kod geliştiricilerin farklı standartlarda kodlama stilleri olmasının zafiyetleri bulmada zorluk çıkardığı tespit edilmiştir.

Büchler vd. (2012), web uygulamalarının ve web servislerinin günümüzde çeşitli saldırılara maruz kaldığından, ayrıca penetrasyon testinin de test uzmanlarının yeteneğine bağlı olduğu için zorluklar yaşandığından bahsetmektedir. Yazarlar çalışmalarında SPACiTE adında bir araç geliştirdiklerinden bahsetmektedirler. Bu yazılım kapsamında genel güvenlik açıklarına karşı potansiyel atak oluşturduğu ve yarı otomatik çalışan sistemde üretilen atakların ne kadarının güvenlik testinden geçebildiğinin raporlandığından bahsetmektedirler. Güvenlik açıkları olarak rol tabanlı erişim kontrolü ve Cross site scripting zafiyeti dikkate

alınmıştır. Ayrıca yazılımın bu açıklara ait atakları başarılı bir şekilde engellemiş olduğundan bahsedilmiştir. İlerleyen çalışmalarda bu aracın web uygulaması olarak geliştirilebileceği fikri yazar tarafından ortaya konmuştur.

Şahinoğlu vd.(2013), test süreçlerinde karşılaşılan problemleri belirli gruplarda toplamışlardır. Probleme neden olan planlama, yönetimsel, gereksinimler ve düşünsel problemler için çözüm önerilerini anlatmışlardır. Proje yönetimiyle ilgili proje, kaynak, zaman planlamasının olduğunu, görev ve test altyapısı ile ilgili ve dökümantasyon eksikliğinden kaynaklanan problemlerden bahsetmiştir. Test yöntemiyle ilgili olarak da yanlış test stratejisi, projenin eksik olması, yetersiz test metrikleri gibi problemlerin varlığından bahsedilmiştir.

3. MATERYAL VE YÖNTEM

Bu bölümde, yazılımların güvenliğini sağlamak adına, projeler kullanıma sürülmeden önce kodları analiz etme yöntemiyle ilgili bilgi verilmektedir. Güvenlik zafiyetlerinin günümüzde yoğun olarak web ortamında yaşanması, tez çalışmasını web uygulama güvenliği konusuna yönlendirmiştir. Bu bölüm kaynak kod analiz, yazılım güvenlik seviyesi belirleme ve yöntem olmak üzere 3 kısma ayrılmıştır. Birinci kısımda, kaynak kodların analizi için statik kod analiz kategorileri ve tekniklerinden bahsedilirken, geliştirilen güvenlik test aracı web uygulamalarını analiz edeceği için web uygulama güvenliğinden ve web zayıflıklarından bahsedilmektedir. İkinci kısımda, yazılım güvenlik seviyesini kod ölçüm parametreleriyle belirleyebilmek adına, kod metriklerinden ve güvenlik seviye tespiti için naive bayes algoritmasından bahsedilmektedir. Son bölümde ise tez çalışmasında kullanılan yöntem belirtilmektedir.

3.1. Statik Kaynak Kod Analizi

Yazılımların geliştirilmesi aşamasında yapılan eklemeler ve değişiklikler, tasarım ve kaynak kod hatalarının zamanla yazılımda yer almasına neden olmaktadır. Bu nedenle, geliştirilen yazılımda oluşan hataların ortaya çıkarılması ve giderilmesi için geliştirme süreci adımlarında statik ve dinamik test teknikleri uygulanmaktadır. Yazılımların çalıştırılmasına gerek kalmadan uygulanabilen statik teknikler arasında kod gözden geçirmeleri, otomatikleştirilmiş statik kod analizi ile birim testleri yer alırken, dinamik teknikler arasında yazılım testleri, sistem entegrasyon testleri ve sistem testleri gibi yöntemler yer almaktadır.

Statik analiz ile çalıştırılmadan analiz edilen yazılımlar için gerekli önlemler, analiz sonuçlarına göre farklı yöntemlerle alınabilmektedir. Statik analiz gözden geçirmeyle veya otomatik analiz araçlarıyla yapılabilmektedir. Statik analiz, araçları daha çok atama ve denetim gibi programlama hatalarını tespit ederken, gözden geçirmeler fonksiyonel hataları da ortaya çıkarabilmektedir (Zheng vd., 2006). Günümüzde proje yönetim süreçlerinin plansızlığı, zaman ve ekonomik maliyet açısından daha uygun olan otomatik kod analiz araçlarının yaygın bir

şekilde kullanılmasına neden olmaktadır. Bazı otomatik analiz araçlarıyla yapılan analizler sadece olası hataları tespit etmekle sınırlı kalmayıp incelenen yazılımın kalite metrikleri, mimari yapısı, kodlama standartlarına uygunluğu gibi önemli özellikleri de ortaya çıkarmaktadır (Kalay, 2009). Statik kod analiz araçları, zafiyet barındıran veya dinamik test uygulaması kolay olan kod parçalarını sürekli kontrol eder. Genellikle kodu inceleyerek, sadece güvenlik sorununun belirtilerinden birini değil sorunların ana sebeplerine işaret ettiği için ve hataları program ilk kez çalıştırılmadan önce geliştirme aşamasında bulabildiği için güvenlik problemlerini tespit etmek için uygun bir yöntemdir. Fakat, statik kod analiz araçları için en sık karşılaşılan şikâyet, güvenlik açıklarının özellikle yanlış alarm olarak bilinen çok fazla yanlış pozitif hata üretmesidir (Chess ve West, 2007).

Statik kod analizi, modern yazılımların geliştirilmesinde önemli bir kıstas olan yazılım kalitesinin arttırılmasında önemli bir rol oynamaktadır. Projelerdeki yazılım hataları veri kaybı gibi ciddi sorunlara yol açabilirken, kaynak kod incelemesi de bu süreçte tek başına yeterli olmamakla beraber yazılım yaşam döngüsünde kendisine yer bulmaktadır. Elle yapılan incelemeler insana bağlı olduğu için bazı hataların gözden kaçmasına neden olabilirken, otomatik kod analiz araçları gözden kaçabilen hataları raporlayabilmektedir.

Statik analiz ile çalışma zamanı hataları, kaynak ve güvenlik sızıntıları gibi birçok farklı hata bulunmaktadır. Bunların yanında, yazılımların metriklerini de hesaplayabilmektedir. Programcılar için programın bakımının kolaylaştırılması ve geliştirilebilmesi amacıyla kodlama standartlarının uygulanması oldukça yararlıdır. Statik analizin diğer bir özelliği ise fonksiyonlar, sınıflar ve dosyalar arasındaki bağımlılıkları hesaplayarak kaynak kod mimarisinin anlaşılmasına yardımcı olmaktır. Statik analizin hassasiyeti harcanan zamana bağlıdır. Daha hassas analiz daha fazla kaynak gerektirir ve zaman alıcıdır. Kusurları bulmak için ne kadar zaman ve kaynak harcanabileceğine karar verebilmeleri, kod geliştiricilerin fikir değiştirmelerine neden olmaktadır. Analizin çok hızlı olması, istenmeyen birçok “yanlış pozitif” hatasını ortaya çıkarmaktadır, fakat aksine analiz çok kapsamlı ise, analizin tamamlanması çok uzun zaman alacaktır (Eklöf,

2011). Statik kod analiz araçları kodla ilgili bilgileri eksiksiz ve doğru çıkarabilmek için kaynak kod bilgisini ve derleme bilgisini kullanmaktadır. Kodun derlenmesi, birçok yanlış pozitif (false-positive) uyarısını önlediği gibi birçok hatanın tespit edilemeden kaçırılmasını (false-negative) da önler (Ademoğlu ve Akyüz, 2008). Yanlış pozitif hatalar, otomatik kod analiz aracının hata olarak tespit ettiği kod satırlarının gözden geçirme yöntemiyle incelendiğinde bulunamamasıdır. Yanlış negatif hatalar, yazılım analiz edilirken açıklıkların statik kod analiz araçları tarafından bulunamamasıdır. Yanlış negatif hatalar yanlış pozitiflerden daha önemli görülebilirken, yanlış pozitif ile yazılımların savunmasız olup olmadığı tespit edilebilir. Yanlış negatif hatalar ise, açıklıklar için genel bir tarama yapılmadığı zaman, statik kod analiz aracıyla ilgili altta yatan bir mantık sorunu olduğunda veya yanlış bir yerde arama yaptığı zaman meydana gelebilir (Dewhurst, 2012).

Son yıllarda, bankalar dâhil birçok kuruluş yazılım ürünlerinin piyasaya sürmeden önce yazılımın statik kod analizini, sızma testini ve güvenlik testini geçmeyi zorunlu hale getirmesi statik kod analizinin önemini göstermektedir. Bu kapsamda, Findbugs ve Fortify gibi statik analiz araçları her geçen gün daha popüler olmaya başlamıştır. Analiz araçları, güvenlik açıklarına veya üretim kalitesi için gerekli diğer kod kalitesi sorunlarına tanımlanan kalıpları arar. Diğer teknolojiler gibi, statik analiz güvenlik analizi alanında en iyi yol olsa da, kendine ait avantaj ve dezavantajları bulunmaktadır. Projedeki takım liderleri ve geliştiriciler projeyi sağlıklı bir durumda tutabilmek ve takımda en iyi uygulamaları geliştirmeyi teşvik etmek için otomatik kod analizini ve otomatik test süreçlerini kurmakla sorumludurlar.

3.1.1. Statik kod analiz kategorileri

Yazılımların yaygınlaşmasıyla birlikte, geliştiriciler tarafından farklı amaçları olan birçok statik kod analiz aracı kullanılmaktadır. Ticari satıcılara bakarak ve uygun olan açık kaynak projeleri incelenerek statik analiz araçlarının farklı kategorileri gösterilebilmektedir. Statik kod analizi; Tip denetimi, stil denetimi,

program anlama, hata bulma ve güvenlik incelemesi olarak 5 farklı kategoride incelenmektedir.

3.1.1.1. Tip denetimi

Statik analizin en yaygın kullanılan biçimidir ve birçok programcı tip kontrolünü, kodu geliştirme aşamasında öğrendikleri kapsamında uygulamaya çalışır. Tip kontrolü programlama hatalarının tüm kategorilerini ortadan kaldırır. Örneğin, programcıların yanlışlıkla nesne değişkenlerine tamamlayıcı değerler atamasını engeller. Derleme zamanında hataları yakalayıp, çalışma zamanında oluşabilecek hataları engeller. Tip kontrolünün hataları yakalamak için belli bir sınırı vardır ve diğer tüm statik analiz teknikleri gibi false pozitive ve false negative üretir. Aşağıdaki kod satırlarında tip kontrolünden başarıyla geçebilen fakat çalışma zamanında kırıma yaşayabilecek bir örnek verilmiştir.

```
Object[] objs = new String[1];
```

```
objs[0] = new Object();
```

Örnekteki java kodu, tip kontrolünden geçecek ve aksama olmadan derlenecektir, fakat çalışma zamanında kırılacaktır. Javada tip kontrolü yapılarak, nesnelerin dizi değişkeninin string değişkene referans tutmasını sağlar (çünkü string sınıfı object sınıfından türetilmiştir), fakat çalışma zamanında java string bir diziye object tipindeki nesneye referans vermesi için izin vermez. Tip kontrolü için örnekteki kod problemsizdir, fakat kod çalıştığı zaman, arraystore exception hatası üretir. Bu tip kontrolüne ait bir false negative dir (Chess ve West, 2007).

3.1.1.2. Stil denetimi

Stil denetimi, genellikle tip kontrolündeki kurallardan daha hassastır. Stil denetimi sadece stil kontrolü; Boşluk, adlandırma, onaylanmayan fonksiyonlar, yorumlama, program yapısı ve benzerleriyle ilgili kuralları uygular. Birçok programcı şiddetle kendi geliştirme tarzına bağlı olduğundan, birçok stil denetleyici yazılımı kurallar hakkında oldukça esnektir. Büyük programlama projelerinde, yarı yolda stil denetimini geliştiricilere kabul ettirmek çok zordur, farklı geliştiriciler farklı kavramları kendileri için doğru stil olarak kabul

etmişlerdir. Statik analiz başladıktan sonra stil denetimi çıktılarını azaltmak için kodun tekrar değerlendirilmesi büyük problemlerin azaltılması için önemli bir fayda sağlar. Java dilinde en önemli stil denetim aracı PMD dir (Chess ve West, 2007).

3.1.1.3. Program anlama

Program anlama araçları büyük kod parçalarının anlamlandırılmasına yardımcı olur. IDE ler en azından bazı “program anlama” işlevselliğini içerir. Basit örnekler "bu metodun tüm kullanımlarını bul" ve "bu global değişkenin açıklamasını bul" gibi işlevsellikleri içerir. Değişkenlerin yeniden adlandırılması veya tek fonksiyonun iki fonksiyona bölünmesi gibi birçok gelişmiş analiz otomatik yeniden düzenleme özelliğini içerir. Yüksek seviye program anlama araçları programların çalışması hakkında programcılara fikir verir. Tersine mühendislik denemeleri programcılara büyük bir pencereden bakmalarını sağlar (Chess ve West, 2007).

3.1.1.4. Hata bulma

Hata bulma aracının amacı ne stil denetleyicisi gibi biçimlendirme sorunlarından şikâyet etmek ne de program doğrulama aracının yaptığı gibi tam ve ayrıntılı bir karşılaştırma yapmaktır. Bunların aksine, hata bulma aracı sadece programcının dikkat etmediği yerleri işaret eder. Hata bulma araçları öncesinde hataları belirten kurallar kümesini oluşturdukları için bu tarz araçların kullanımı basittir. Bazı hata bulma araçları özellik denetim araçlarının kullanacağı aynı tür algoritmaları kullanırlar. Fakat bu yanlış negatif sayılarının yüksek olacağı anlamına gelse de hata bulma araçları genellikle düşük seviyede yanlış positif ler üretir (Chess ve West, 2007).

3.1.1.5. Güvenlik İncelemesi

Modern güvenlik araçları, özellik denetleyicileri ve hata bulucuların birleşimidir. Birçok güvenlik özelliği kısaca program özelliği olarak ifade edilebilir. Özellik

denetimi için, potansiyel tampon taşma açığını aramak "Program, ayrılan bellek sınırı dışında bir adrese erişemez" özelliğini kontrol meselesi olabilir. Hata bulma etki alanından itibaren güvenlik araçları, geliştiricilerin problemleri çözmek için güvensiz programlama deyimi olarak bilinen aynı güvensiz yöntemleri yeniden keşfetmeye devam etme kavramını benimser. Güvenlik araçları genellikle hata bulma araçlarının "yanlış negatif" lere izin verme pahasına "yanlış pozitif" leri minimize özelliğini devralmaz. Güvenlik araçları dikkatli olsalar bile hata eğilimindedirler ve araçlar hatayı tespit edemeseler bile projeler manuel incelenebilir olmalıdır (Chess ve West, 2007).

3.1.2. Statik analiz yöntemleri

Statik analizin amacı, kusurlu verilerin temizlenmeden uygun hale gelip gelemeyeceğini belirlemektir. Yazılımlardaki kusurlu verilerin temizlenmeden uygun hale gelip gelemeyeceğini belirlemek amacıyla fonksiyonlarda kullanılan değişkenlerin hatalı değerlerini belirlemek için bilgisayar bilimlerinde iyi bilinen ve son yıllarda derleyici optimizasyonu alanında da kullanılan veri akış analizi yönetimi kullanılır (Kalay, 2009; Yılmaz, 2009; Särökaari, 2012). Veri akış analizinin amacı her bir program parçasını (fonksiyonlar gibi büyük parçalar) istatistiksel olarak hesaplamaktır. Klasik sabit analizi ise her bir program parçası için değişkenlerin tuttuğu gerçek değerleri hesaplar. Statik analiz genel olarak; Veri akış analizi, sabit analizi ve tip çıkarımı yöntemleriyle analiz edilmektedir.

3.1.2.1. Tip çıkarımı

Programlama dilinin tip sistemi, tip tanımı ve kurallarını, tip kapsayıcılığı, tip bağılılığı ve tip denkliği için sonuçlandırır. Tip bağılılığı, bir program içindeki değişken ve fonksiyonların tipini, gelen nesnelerin uygun olup olmadığını belirlemek için otomatik olarak üretir. Bu tür bağılılıklar tiplerdeki açıkları incelemek için kullanılabilir ve gerekli tip dönüşümlerini uygun işlemlerle yürütür. Bu, yazılımdaki güvenlik tehditlerini daha hızlı tespit etmeyi sağlayan yüksek verimliliğe ve basitliğe sahiptir. Tip çıkarımı "format string" açıklarının

tespitinde, işletim sistemi çekirdeğinde savunmasız işaretçi kullanımlarında uygulanır (Satyanarayana, 2009).

3.1.2.2. Veri akış analizi

Programlardan anlamlı veriler toplayan, sonrasında bunları tanımlayan ve değişkenleri cebirsel yaklaşımla kullanan veri akış analizi süreç programlamada kullanılır. Program optimizasyonunda, program doğrulamasında, hata gidermede, paralel, vektörel ve seri programlama çevresinde kullanılır (Satyanarayana, 2009).

Veri akış analizi, programdaki veriler hakkında onları çalıştırmadan çalışma zamanı bilgilerini almayı sağlayan bir süreçtir. Statik aşamadayken veri hakkında çalışma zamanı hatalarını toplar. (Wögerer, 2005). Bununla birlikte, veri akış analizi, operatörlerin anlamlarından faydalanmazken, programlama dili söz diziminin anlamsallığı, algoritmaların kurallarından çıkarılır (Wögerer, 2005). Veri Akış analizi, programların veri-akış analizini gerçekleştirmek için basit bir kontrol akış diyagramıdır. Her düğüm için veri-akış denklemleri kurmak ve tüm sistemi dengelemek için tekrar tekrar her düğüm için girdi çıktı hesaplayarak sistemi dolaşmaktadır. Analiz, bir işlemin her noktasında özel bilgi elde etmek için çalışır. Genellikle, temel blok noktalarında bilgileri hesaplamak daha kolay olduğu için, temel blok sınırları bu bilgiyi elde etmek için yeterlidir. Veri akış analizinde 3 genel terim bulunmaktadır. Bunlar temel blok, kontrol akış grafiği ve kontrol akış dizinidir (Dewhurst, 2012).

3.1.2.3. Kusur analizi

Kusur analizi, kullanıcı girişi bağımlılık denetleme olarak adlandırılmaktadır. Kusur analizinin temel amacı, kullanıcı tarafından (doğrudan veya dolaylı) değiştirilebilir herhangi bir değişkenin güvenlik açığı oluşturup oluşturmayacağına bakmaktır. Kullanıcı kontrol girişi ile 'kusurlu' olan hassas fonksiyonlar izlenmektedir. Kusurlu değişken temizleme işleminden geçmediği takdirde güvenlik açığı olarak belirlenir (Karayumak, 2013). Potansiyel

savunmasız fonksiyonlar kaynak kodun içerisinde dize eşleştirme yoluyla tanımlanan “savunmasız fonksiyon (sinks)” olarak bilinir. Potansiyel savunmasız fonksiyonlara gönderilen parametreler ilk andan itibaren veri akış analizi ile takip edilir. Kullanıcı kontrolünde girilen fonksiyonların parametreleri kusurlu ise, kaynak kodlar incelendiğinde savunmasız olarak belirtilir. Bununla birlikte kullanıcı kusurlu değişkenler güvenli fonksiyonlar ile güvence altına alınarak, kaynak kod savunmasız kategorisinden çıkarılır (Dahse, 2010).

Perl ve Ruby gibi bazı programlama dilleri, arayüz vasıtasıyla gelen veriler gibi bazı durumlarda kodun kusur kontrolünden geçirilmesini sağlar. Aşağıdaki kod satırlarında Php ile temel kusur analizine ait birinci kod bloğu gösterilmektedir.

```
$user_tainted = $_GET['tainted'];  
echo($user_tainted);
```

Yukardaki birinci örnekte kullanıcı tarafından sağlanan \$user_tainted değişkeni global \$_GET değişkeni vasıtasıyla verilir. Kullanıcının girdiği \$user_tainted değişkeni daha sonra sisteme çıktı veren echo() koduna aktarılır. Bu XSS açığına verilebilecek basit bir örnektir.

```
$user_tainted = $_GET['tainted'];  
$sanitized = htmlspecialchars($user_tainted);  
echo($sanitized);
```

Son örnek, \$user_tainted değişkeninin arındırma fonksiyonundan, HTML özel karakterlerden geçirilmesinin ve XSS e karşı savunmasız olmayan kod parçaları dışında, ilk örneğin benzeridir. Htmlespecialchars fonksiyonu Çizelge 3.1’ deki özel karakterleri HTML deki denkliğine dönüştürür (Dewhurst, 2012).

Çizelge 3.1. Özel karakterin html denklikleri

Özel Karakter	HTML
&	&
“	"
‘	'
<	<
>	>

Çizelge 3.2’ de örnek kod satırının htmlspecialchars fonksiyonundan geçirildikten sonraki çıktısı gösterilmektedir.

Çizelge 3.2. Örnek kodun htmlspecialchars fonksiyonundan geçirilmesi

Fonksiyon Öncesi	<script>alert(1)</script>
Fonksiyon Sonrası	<script<alert(1)</script>

Kusur analizinde, potansiyel açıklıkları tespit ve temizleme aşamalarında kaynaklar, savunmasız fonksiyonlar ve temizleme fonksiyonları olmak üzere 3 veri dizisi kullanılır. Kaynaklar; Kullanıcılar, dosyalar, veritabanları veya diğer kullanıcı kontrollü girdilerden elde edilen muhtemel kirli verilerdir.

Php için kullanıcı bağımlı kusur fonksiyonları \$Globals, \$_Server, \$_Get, \$_Post, \$_Files, \$_Cookie, \$_Session, \$_Request, \$_Env olarak bilinmektedir. Dosya bağımlı kusur fonksiyonları ise file(), fopen(), popen(), file_get_contents(), fread(), fscanf() dir. Veritabanı bağımlı kusur fonksiyonları olarak mysql_fetch_array(), mysql_fetch_assoc(), mysql_fetch_field(), mysql_fetch_object(), mysql_fetch_row() kullanılmaktadır.

Savunmasız Fonksiyonlar (Sinks), SQL sorguları çalıştırma veya veri çıktısı gibi kaynakların sona erebildiği potansiyel hassas fonksiyonlardır. echo(), print(), printf() fonksiyonları XSS, mysql_query(), pg_query() SQL Injection, exec(), shell_exec(), system(), proc_open() komut işletimi, include(), require(), require_once(), include_once() ise dosya sızma saldırılarına neden olabilmektedir.

Temizleme fonksiyonları savunmasız fonksiyonlara erişilmeden önce zafiyet barındıran değişkenlerin kaynağını temizler. XSS zafiyetlerini engellemek için htmlspecialchars(), htmlentities(), SQLI için mysql_real_escape_string(), sqlite_escape_string(), Komut işletimi için escapeshellcmd(), escapeshellarg() fonksiyonları kullanılmaktadır (Dewhurst, 2012).

3.1.2.4. Sözcüksel analiz

Sözcük analizi, kaynak kodu daha kolay işlemek adına simgesel durumdan anlamlı bilgi haline getirir. Analiz kodu alt parçalara ayırmaktadır. Php

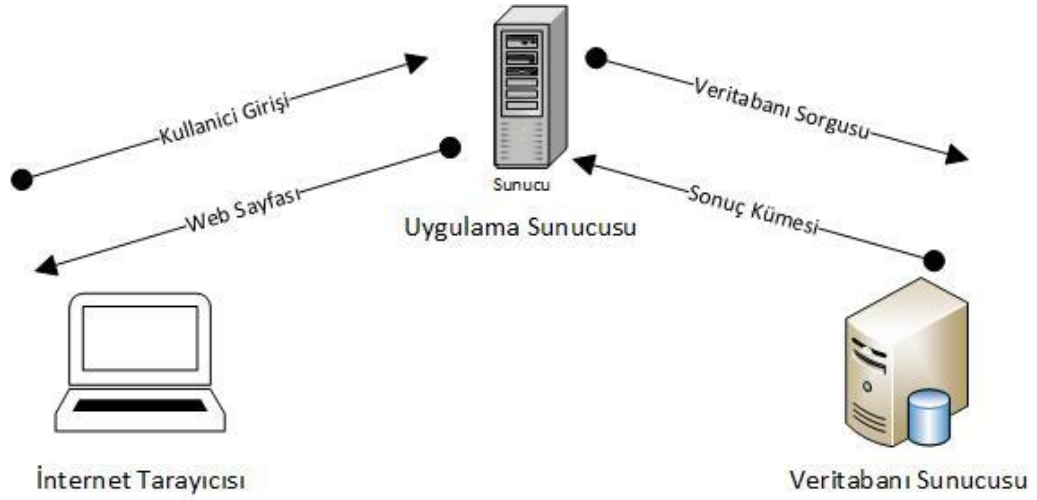
programlama dilinde analiz için token_get_all ve token_name fonksiyonları kullanılır (Dewhurst, 2012).

Statik kod analizinin avantajları yanında, bazı kısıtları da bulunmaktadır. Örneğin statik kod analizi mantıksal hataları tespit edemez. Statik kod analiz araçları programın bütününe bilmediği için, kaynak kodunu analiz ederek bireysel kullanıcılar için kullanılabilir olması gereken işlevsellikten habersizdirler. Örneğin statik kod analiz araçları bireysel bir kullanıcının veritabanı değişikliği yapıp yapmayacağını bilemez. Diğer bir statik kod analiz kısıtı ise yanlış negatif ve yanlış pozitiflerdir. Statik kod analiz araçları geliştiricilerinin yazmış olduğu kurallarla sınırlıdır; Bu kurallar verilen modele göre çok sınırlı veya çok geniş çaplı olabilmektedir (Dewhurst, 2012).

3.2. Web Uygulama Güvenliği

Web uygulamaları; Günümüz online bankacılığı, alışveriş, üniversite bilgi sistemleri, eposta, sosyal ağ ve çeşitli devlet işlerinde kullanılmaktadır. Her yerden erişilebilen web teknolojisi sayesinde kullanıcılar dinamik olarak üretilen içeriklere erişebilmek için sadece web tarayıcısına ve internet bağlantısına ihtiyaç duymaktadırlar. Kredi kartı veya ürün stok bilgileri benzeri gizlilik gerektiren web verileri hem kullanıcılar hem de servis sağlayıcılar için önemli bir değere sahiptir. Kullanıcılar gezindikleri web sitelerinin güvenli olduklarına inanırlar çünkü web tarayıcıları kullanıcıların local bilgisayarlarında çalışmaktadır fakat bazı sitelerde zafiyet barındıran yazılım hataları bulunmaktadır.

Web uygulama sunucuları bilgi girişlerini form yardımıyla alır; kullanıcı girişleri tarayıcıdan girilirken sonuçlar ise veritabanından gelmektedir. Web uygulama kodları sunucularda dinamik olarak üretildiği için web uygulamasını oluşturan kodlar herhangi bir yerde bulunmaz (Wasserman, 2008). Şekil 3.1' de web uygulamalarının genel bir çalışma yapısı gösterilmektedir.



Şekil 3.1. Web uygulama mimarisi

Web uygulamalarının birçoğu kritik iş uygulamaları için kullanılmaktadır ve kullanıcıların kişisel, finansal verilerini saklamaktadırlar. Saldırganlar yeni açıklar için sistemleri sürekli analiz ederler ve bulguları kötüye kullanmak için açıklar oluştururlar. Site sahiplerinin bulunan açıklıklara karşı korunması kritiktir. Saldırganın hassas verileri ortaya çıkarması ve sınırsız bir şekilde sisteme erişebilmesi sitenin ait olduğu firmanın iş ve itibarı üzerinde ciddi bir etkisi olabilir. Web uygulama güvenliği son birkaç yılda büyük bir etki yaratmıştır. Anonymous ve LulSec gibi gruplar çok sayıda kamu ve özel kuruluşa saldırmış ve onlardan aldığı veriyi kamuoyuna sızdırmıştır. Aynı zamanda, milyonlarca kullanıcının şifreleri çalındığı ve sızdırıldığı LinkedIn kazası, web uygulama güvenliğinin önemi ve firmaların açıklara karşı korunmalarına verilebilecek birçok örnekten birisidir (Särökaari, 2012).

3.2.1. Web Uygulamalarının zayıflık tespiti

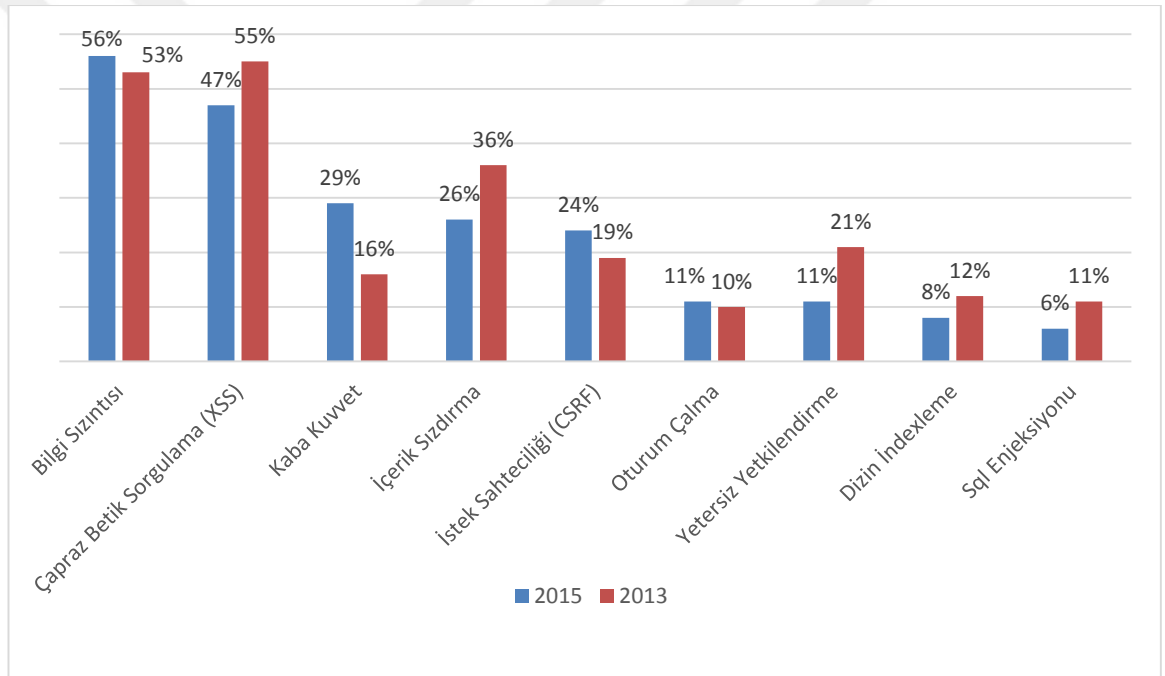
Bir dilin ya da platformun güvenlik analizi yapılırken genel olarak; Platformun kendisinden çıkan açıklar, platform tarafından geliştiriciye sunulan güvenlik kütüphaneleri, güvenlik önlemleri almak için yapılması gereken işlemlerin kolay olması ve varsayılan ayarlar kriter olarak gözetilmektedir. Geliştirici, ilgili kriterlere göre platform ve dil seçimini yaptığı takdirde uygulamanın güvenliğini büyük ölçüde zahmetsiz bir şekilde sağlamış olacaktır.

Proje yöneticisinin veya geliştiricinin; web uygulamalarına yönelik saldırılar ve web uygulamaları güvenliği konusunda bilgisiz olması, uygulamalarda oluşan hataların en büyük sebebidir. Uygulamanın geliştirilme sürecinde uygulamaya yazılımsal olarak katkı sağlayan geliştiriciler veya proje yöneticilerinin uygulamaya ne tür saldırılar gelebileceği konusunda bilgisiz olması, saldırının nerelerden gerçekleşebileceğini görememesi gibi nedenlerden ötürü uygulamalar saldırılara açık hale gelmektedir. Geliştirme sürecine katkıda bulunan kişilerde güvenlik farkındalığının oluşması, saldırılar konusunda bilgi edinilmesi ve uygulamanın bu donelere göre revize edilip güvenli hale getirilmesi sorunları büyük ölçüde çözmektedir. Yeni teknolojilerin geliştirilmesine paralel olarak bu teknolojilere zarar vermek amaçlı geliştirilen yeni tekniklerin bilinmemesi, web güvenliği sektöründeki gelişmelerin takip edilmemesi, geliştirme sürecinden sonra uygulamaya gelebilecek saldırıları açıklamaktadır.

Müşterilerin taleplerine göre projenin çoğu zaman yeniden şekillenmesi, bu süreç esnasında yapının bozulması ve güvenlik açıklarının oluşması, uygulamaya yönelik gelebilecek saldırıların niceliğini ve niteliğini doğrudan etkilemektedir. Standart bir geliştirme döngüsünün izlenmesi, bu sorunu ortadan kaldırmaya yardımcı olacaktır. Projenin geliştirme ve test süreçlerinin yeterince uzun olmaması nedeniyle yapıda bozukluklar ortaya çıkmaktadır. İyi bir proje ve zaman yönetiminin uygulanması, her zaman için faydalı olmaktadır. Projeler ilgili alanda uzmanlaşmış kişi ya da kurumlara yaptırabilmektedir. Dışardan bir firmaya yaptırılan yazılımların birçok avantajı olsa da uygulamayı geliştiren kişilerce kötü niyet gözetilerek uygulamada güvenlik açıkları bırakılabilmektedir. Güvenlik açıklarını tespit ederek tüm dünyaya tanıtan kuruluşlar sayesinde yeni saldırılar hakkında bilgi ve korunma yöntemleri öğrenilebilmektedir. Bu kuruluşlar arasında yerini alan OWASP, güvenlik alanında lider bir gönüllü kuruluştur. OWASP belirli dönemlerde güvenlik raporları açıklayarak web ortamındaki saldırılar hakkında kullanıcı ve geliştiricilere bilgiler sunmaktadır. Güvenlik konulu makaleler, dökümanlar ve projeleri yayınlayarak, web uygulamalarının ortaya çıkardığı zararları en aza indirme yolunda çalışmalar yapmaktadır. Dünyada yapılan web uygulaması güvenliği konulu çalışmaların

takibini sağlayarak web tabanlı saldırıları en aza indirmeye çalışan OWASP 2013 yılında yayınladığı rapora göre web uygulamalarındaki zafiyetler sırasıyla Sql enjeksiyonu, oturum yönetimi, XSS, dosya erişimi, sunucu veya veritabanındaki güvenliğin ayarlanmaması, hassas veri, siteler arası istek sahteciliği (CSRF), güvensiz yönlendirmeler olarak listelenmektedir (Owasp, 2013).

Web uygulamalarındaki güvenlik açıklarının hedef sitelerde bulunma oranları ise WhiteHat Security şirketinin (WhiteHat Security, 2015) yapmış olduğu araştırmaya göre 2012 ve 2015 yıllarında şekil 3.2' deki gibidir.



Şekil 3.2. Güvenlik açıklarının yıllara göre web uygulamalarında görülme ihtimali

3.2.2. Web uygulama açıklıkları

Web uygulama açıklıkları yetkilendirme, şifreleme, bilgi sızıntısı ve girdi doğrulama kategorilerinde incelenmektedir.

3.2.2.1. Yetkilendirme

Yetkilendirme problemlerinin 2010 yılından 2012 yılına kadar artış göstermiştir. Karakter doğrulama (Captcha) kullanımının yaygınlaşmasıyla şifre çalma açıkları yıldan yıla azalırken, bununla birlikte uygulamalardaki ilk kullanıcı girişlerinde şifre resetleme ve şifre değiştirme açıklıkları, yıldan yıla artış göstermiştir. Ortalama, 2010 ve 2012 de uygulama başına ortalama 1,8 birim açıklık tespit edilirken, 2011 de test edilen web uygulamalarında 2 birimin üzerinde yetkilendirmeyle ilişkili açıklıklar tespit edilmiştir. Web uygulamalarının kimlik doğrulamasını etkileyen en yaygın sorunlardan iki tanesi eş zamanlı oturum doğrulaması (uygulamaların %50 sini etkiler) ve zayıf şifre politikasıdır(%47) (Tudor, 2013). Zayıf şifre politikaları, kaba kuvvet (brute force) korumasının olmaması, şifre değiştirme mekanizmalarının zayıf olması, zayıf kullanıcı adı kalitesi, fonksiyonel çıkış işleminin olmaması, ilk girişte şifre değişiminin yaptırılmaması, captcha kullanılmaması, eş zamanlı kimlik doğrulamaya izin verilmesi ve yetkilendirme atlatma açıklıkları yetkilendirmeyi aşarak zafiyet oluşturan nedenlerdir.

3.2.2.2. Şifreleme

Şifreleme açıklığı 2012 de 0,9 birime inmişken hem 2010 hem de 2011 de uygulama başına 1 birim şifreleme açıklığı bulunmuştur. Hassas verilerin şifrelenmemiş veri iletimiyle transfer edilmesinin web uygulamalarındaki oranı 2010 ve 2011 yıllarında 30% civarındayken 2012 de %23 e düşmüştür. Şifrelenmemiş veri iletimi, zayıf SSL şifreleri, zayıf giriş katmanı (SSL), geçersiz dijital sertifika, geçersiz dijital sertifika, zayıf şifreleme algoritmaları şifreleme açıklıklarının bulunduğu uygulamalardır (Tudor, 2013).

3.2.2.3. Bilgi sızıntısı

Web uygulamalarının büyük bir kısmında bilgi sızıntısı bulunmaktadır. 2010' da test edilen web uygulamalarında ortalama 2.2 birim olarak bulunan bilgi sızıntısı açığı 2011 için 2.5 birim ve 2012 için 2.1 birim olarak bulunmuştur. Sızıntı terimi

sunucu yazılımı yerine uygulama bazlı bilgi sızıntısını tarif eder (Böyle durumlarda sunucu konfigrasyon sorunları olarak kategorize edilir). En yaygın sızıntı açıklığı, kullanıcı hata mesajlarının ortaya çıkmasıdır. Genel karşılaşılan örnek ise giriş sayfası hata mesajı gönderdiğinde, geçerli bir kullanıcı bulunamadı hatasıdır. Saldırgan geçerli bir kullanıcı adı bulmak için bu bilgiyi istismar edebilir. Uygulamaların büyük bir kısmını etkileyen diğer sızıntı açıkları ise paylaşılan bir bilgisayar ortamında önemli olabilecek bir numaranın hassas form alanları için otomatik tamamlayıcıları elimine etmesi için sisteme eklenmesidir (Särökaari, 2012). Kullanıcı hata mesajlarının ortaya çıkarılması, otomatik tamamlamanın devre dışı bırakılmaması, Cevap verilerinde hassas bilgiler, hassas içeriğin sayfayı ön belleğe alması, hassas dosya ve izin keşfi, hassas isteklerin ön belleğe alınması, URL de hassas bilgi ve şifresiz görüntü durumu (viewstate), şifrelenmemiş görüntü durumu (Viewstate) bilgi sızıntısı bulunduran web uygulamalarıdır.

3.2.2.4. Girdi doğrulama

Girdi doğrulama başka bir kullanıcı tarafından sağlanan herhangi bir giriş için doğru test yöntemidir. Kullanıcı girişleri bir son kullanıcı, başka bir uygulamaya, kötü niyetli bir kullanıcı gibi çeşitli kaynaklardan gelmektedir. Kötü niyetli bir kullanıcı yazılımlara saldırı yaparken hiçbir bilgi vermez. Web uygulamalarına kimin veya neyin giriş olarak ne gönderdiği bilinmemesinden dolayı tüm girişler için kontrol ve doğrulama yapılmalıdır. Uygulamalardaki muhtemel girişler veritabanından, uygulamadan veya diğer uygulamalardan gelebilmektedir. Bu sebeple tüm girdileri kontrol etmek yazılım uygulamalarındaki potansiyel güvenlik açıklarına karşı koruma olabilmektedir.

Problemler hatalı giriş doğrulamasından kaynaklanmaktadır. Bölüm 3.2.3' de girdileri doğrulama yaparak çözülebilecek bazı açıklar listelenmektedir (Testingsecurity.com, 2014).

3.2.3. Web uygulama saldırı tipleri

Web uygulama saldırıları 3.2.2 de belirtildiği gibi 4 kategoride incelenmektedir. En yaygın saldırı tipleri girdi doğrulama ile ilgili olan saldırılardır. SQLI, XSS, dizin gezinimi, dizin servis standartı (LDAP) ve javascript enjeksiyonu girdi doğrulamanın eksikliği sonucu oluşabilen saldırılardır. Bunun yanında bellek taşması, hizmet engelleme (DOS) ve dağıtık hizmet engelleme (DDOS) saldırıları da web uygulamalarına zarar verebilmektedir

3.2.3.1. Sql Enjeksiyonu

SQL enjeksiyonu uygulamalara güvenilir olmayan kaynaklardan gelen girdilerin kontrol edilmeden SQL sorgusunu oluşturan cümlecik içerisinde kullanılmasından kaynaklanmaktadır. Modern web uygulamalarının hemen hemen hepsi ilişkisel veritabanı ve yapısal sorgulama dilini kullanmaktadır. Fakat, gerek kullanılan yöntemler gerekse de kullanılan uygulama çatıları nedeni ile web teknolojisinin ilk yıllarındaki kadar her uygulamada görülebilecek bir zafiyet olmaktan çıkmıştır. Her uygulamada olmasa bile, hala azımsanmayacak sayıda uygulamada SQL enjeksiyonu zafiyetine rastlamak mümkündür.

SQL dili, uygulamaların arka uçta bulunan veritabanı sunucuları ile konuşmasına olanak sağlayan bir dildir. Uygulama geliştiriciler kullanıcıların uygulama üzerinden göndermiş oldukları verileri işleyerek veritabanına kaydeder ve daha sonra bu verilerin bir kısmını da gösterim için kullanırlar.

“www.ornekadres.com/sinif.aspx?class=1A” Uygulama isteğinin SQL dilindeki karşılığı “Select name from sinif where class='1A'” olarak görülmektedir. SQLI bu sorguların manipüle edilmesi sonucunda veritabanından dönen cevapların farklılaştırılması ile fark edilir. Eğer sorguda 1A yerine 1B öğrencilerini getir ifadesi yazıyorsa sadece bir sınıfın öğrencilerini getirme öngörüsü manipüle edilmiş olur. Sızma testi yapanlar açısından SQLI uygulamanın birçok yerinde kullanıldığı için uygun bir sızma yöntemidir. Güvenli kod geliştirme prensiplerine uygun bir şekilde uygulamalar geliştirilse de çok sayıda girdi kontrolünün

uygulanabilir olmasının zorlukları sebebiyle herhangi bir SQLi zafiyetine sahip olan uygulamanın güvensiz olduğu belirtilmektedir. Aşağıdaki örnek kod parçası klasik SQLi' na örnek olarak gösterilmektedir.

```
String sinif=txt_sinif.Text;  
String sorgu="SELECT * FROM okul WHERE sinifkod="+sinif+";  
SqlDataAdapter da=new SqlDataAdapter(sorgu,connection);  
DataTable dt=new DataTable();  
da.Fill(dt);
```

Yukarıdaki kod bloğunda sinif parametresi hiç bir kontrolden geçirilmediği için SQLi zafiyeti yaratılabilen bir kod oluşturulmuştur. SQLi tespiti yapılan isteğe karşın sunucudan alınan cevabın analiziyle mümkün olmaktadır. Verilen cevapların farklılıkları zafiyetleri kategorilerine ayrılmaktadır.

Klasik SQLi' da "where sinifkod='1' or '1'='1'" yazıldığında her zaman true dönecektir. Yani tablodaki tüm verilerin dönmesini sağlayacaktır. Bu klasik SQLi oluşturulmuş olduğu anlamına gelmektedir. Normal şartlarda sinifkod parametresi 1 veya 1A parametresini beklemektedir. Bunun yerine ' tırnak değeri gönderildiğinde uygulama yazım hatası veriyor olacaktır. Hata mesajları da SQLi zafiyet tespitinde kullanılabilmektedir.

Union tabanlı zafiyet ise, iki veritabanı tablosunun birleştirilmesiyle oluşmaktadır. Saldırıların tespiti klasik SQLi saldırılarının tespiti ile aynıdır. Saldırı sonucunda alınan hatalar kolon sayılarının aynı olması gerektiğidir. İstismar için kullanılacak kolon sayısının bulunması, SQL sorgularındaki ORDER BY kelimesi, kolon sayısının bulunmasını sağlayacaktır. "Select * from okul ORDER BY 1" Sorgusu ile sinifkod kolonuna göre sıralama yapılabilmektedir. "ORDER BY 3" Sorgusu ile sinifad kolonuna göre sıralama yapılabilmektedir. "ORDER BY 4" İle sorgu hata vermekte ve bu da tablonun 3 alandan oluştuğunu belirtmektedir. "www.ornekadres.com/sinif.aspx?class=1A" İle 1.sınıf öğrencileri listelenmiş olur. "www.ornekadres.com/sinif.aspx?class=1A union select 1-- " Sorgusu manipulasyon için örnek kod satırıdır (Demir, 2013).

“www.ornekadres.com/sinif.aspx?class=1A’union select null,null,null—“ İsteği sonucunda, tablodaki tüm veriler listelenmiş olur. Ayrıca alanların veri tipleri de sorgularla bulunabilmektedir. Adres satırına “select 1,2,@@version –“ yazıldığı takdirde veritabanı versiyonu da bulunmuş olur.

Hata tabanlı SQLI, uygulama geliştiricilerin veritabanı hata mesajlarını aleni bir şekilde kullanıcılarına göstermesi sonucu oluşmaktadır. Elde edilen hata mesajları sayesinde kullanılan veritabanı adı ve türü, tablo adı, veri tipi ve uzunluğu, metodlar vb. gibi parametreler hakkında bilgi elde edilmekte ve bu bilgiler doğrultusunda hedefe daha az adet deneme ile ulaşmak mümkün olabilmektedir. Bazı uygulamalarda hata mesajları kullanıcıların göreceği şekilde olmayıp yorum satırı içerisinde saklanabilmektedir. Dolayısıyla yapılan isteğin cevabı tarayıcıda hata mesajı olmaksızın gözüke de hata mesajının HTML kaynak kodu içerisinde olabileceği, bu nedenle yapılan isteğe karşın gelen cevabın HTML kaynak kodlarının incelenmesi gerektiği unutulmamalıdır. Adres satırına tek tırnak eklendiği zaman “Unclosed quotation mark after the character string”, tektırnak yerine string bir değer girildiği zaman “Error converting data type nvarchar to int” hata mesajı alınarak tam sayı beklenen bir alan olduğu tespit edilebilmektedir. Veritabanı sunucusu hakkında bilgi almak için adres satırına “SELECT a” yazılabilir. Bu şekilde kolon isimleri hakkında bilgi alınabilir. Adres satırına “SELECT convert(int, @@version)” yazıldığında ise veritabanı sürüm bilgisi elde edilir. “SELECT convert(int,user)” v.b. sorgular yazılarak istenilen veriler çekilebilmektedir.

Kör SQLI, uygulamadaki hata mesajlarına bakmak yerine zafiyet barındıran sayfanın manipülasyonlara verdiği cevapların doğru/yanlış olması ile ilgilenir. Son zamanlarda uygulamalar yaygın olarak hata mesajlarını ayrıntılı olarak kullanıcıya göstermek yerine özelleştirilmiş bir hata mesajı veya sayfası dönmeyi tercih etmektedirler. “www.ornekadres.com/sinif.aspx?class=1 sınıfı=1” olan kişi bilgilerini getirir. class=1 AND 1=1 isteğinde eğer istenen sayfa görüntüleniyorsa class=1 AND 2=1 sorgusunda bilgiler getirilemiyorsa kör SQLI zafiyeti vardır. Veya class=1’ de istenen sonuçlar gelirken class=1-1 ile de aynı sonuç görüntüleniyorsa bu zafiyetten bahsedilebilir. SELECT database() sorgusu

proje veritabanı ismini verir. "SELECT substring((SELECT database()),1,1)" Veritabanı adının ilk harfini verir.

"www.ornekadres.com/sinif.aspx?class=1 AND ASCII (SUBSTRING((SELECT database()),1,1))=97" İsteği sunucuya gönderildiğinde veritabanında class=1 olan ve 100=97 mi sorgusu çalışacaktır. 100=97 eşitliği sağlanamadığı için "yanlış" dönecektir. Sırasıyla b,c ve diğer harflerin ascii karşılığı denendiğinde doğru harf bulunacaktır. 2. harf ve diğerleri için tek tek bu işlemler gerçekleştirilerek doğru veritabanı ismi bulunabilecektir.

Zaman tabanlı SQLi saldırılarında yapılan isteğe karşın sunucudan dönen cevabın içeriği yerine cevabın dönüş süresi göz önünde bulundurularak zafiyetin varlığı veya yokluğundan söz edilmektedir. "www.ornekadres.com/sinif.aspx?class=1A';WAITFOR DELAY '0:0:10';--" Adresi sunucuya gönderildiğinde 10 sn bekleme sonunda listeyi getirdiği takdirde zafiyetin bulunduğu söylenebilmektedir. Kör SQLi' da olduğu gibi aynı istismar yöntemi burda da kullanılabilir.

SQLi zafiyetinin önlemleri arasında da uygulamanın positif girdi denetimi yapması tavsiye edilmektedir. Positif girdi denetimi yaklaşımı ile birlikte SQLi zafiyetlerinde kesin çözüm olarak, bazı programlama dillerinin hazırladığı fonksiyonlar kullanılmalıdır (Demir, 2013)

3.2.3.2. Çapraz Site Betiklemesi

Siteler arası sorgu çalıştırma zafiyeti, web uygulamalarında sıklıkla karşılaşılabilen zafiyetlerden biridir. Yapılan sızma testlerinde test edilen uygulamaların %70-80 kadarında bu zafiyet görülmektedir. Yayınlanan güvenlik araştırmalarında da bulunma sıklığı günümüz web uygulamalarında devam etmektedir. Bununla birlikte her geçen gün gerek uygulama geliştirme ortamları araç ve yöntemlerinin gelişmesi gerekse de önceden fark edilmemiş farklı saldırı türlerini görmek mümkün olmaktadır. XSS saldırıları HTML, stil katmanları (CSS), javascript gibi dillerle hazırlanmış kod parçacıklarının hedef kullanıcının istemcisinde izinsiz olarak çalıştırılmasından kaynaklanmaktadır. XSS istemcide

sonuçlarsada aslında bir uygulama problemidir. Çünkü uygulamanın kötü niyetli kullanıcılardan aldığı veriyi herhangi bir güvenlik kontrolüne tabi tutmadan diğer kullanıcılara göndermesinden kaynaklanmaktadır. Bir uygulamanın XSS zafiyeti barındırması saldırgana, kurbanın oturum bilgisinin çalınmasına, kurbanın o gün görmekte olduğu uygulamanın içeriğinin değiştirilmesine, kurbanın tuş girişleri, ekran görüntüsü vb. aksiyonlarının elde edilmesine, olta saldırılarının gerçekleştirmesine, kurbanın o an uygulamayan saldırganın istediği bir girdiyi veya işlevi eklemeye/tetiklemeye vb. aksiyonlar alınmasına izin verebilmektedir. Herhangi bir kurumda kullanıcının hesap bilgilerinin çalınması gibi yüksek etkili saldırılar meydana gelebilir. XSS zafiyeti yansıyan, depolanmış ve doküman tabanlı olmak üzere 3 kategoride incelemektedir.

Yansıtılmış XSS zafiyetinde, günümüz uygulamaları dinamik sayfaların tekrar kullanıcılara gösterilmesine olanak sağlamaktadır. Bir sayfadaki arama kutucuğuna bir kriter girildiğinde dönen sonuçlar arasında istenilen kriter görünüyorsa bu yansıtılmış bir örnek olarak gösterilebilir. Arama kriteri yerine “<script>alert(güncel)</script>” yazıldığında ve herhangi bir kontrolden geçmediğinde alert metodu çalıştırılacaktır. Bu örnek klasik XSS örneğidir. “www.ornekadres.com/sinif.aspx?mesaj=<script>alert(güncel)</script>” cümleciği XSS saldırısı için örnek bir denemedir. Oturum bilgisini almak için de script etiketleri arasına yazılan kodlar yol gösterici olmaktadır.

Depolanmış XSS zafiyetinde, saldırgan hazırlamış olduğu zararlı kod parçacığını uygulamaya gönderir ve uygulama bu kötü niyetli veriyi veritabanına veya dosya sistemine kaydeder. Zafiyeti anlamak için kayıt edilen bir veriyi kullanıcıya gösterirken yetki girişli sitelerdeki yorumları gibi alanlarda ise tek bir kullanıcıya, ürünlerin yorum satırı ise yorumu görmek isteyen tüm kullanıcılar potansiyel depolanmış saldırı adayı olabilmektedirler. Yansıtılmış XSS den farkı kayıt edilmesiyle ortaya çıkmasıdır. Kullanıcıların isteğine bağlı olmadığı için depolanmış XSS, yansıtılmış XSS e göre daha fazladır çünkü kullanıcıların sadece sayfaya erişiyor olması yeterli olacaktır. Bu sayfalar genellikle profil, yorum veya mesaj sayfaları olmaktadır. Saldırgan veritabanına kaydettirdiği kod parçacığı ile istekte bulunan kurbanların oturum bilgilerini kendisine ait sunucuya

gönderebilir. Saldırgan oturum bilgileriyle kurbanların adına isteklerde bulunabilmektedir. Tespit etmek için, isteği yapılan verinin kaydedildiği alanın görüntülenmesi yeterlidir.

Sosyal mühendisliğe dayanan bir saldırı çeşiti olan oltalama, kurbanları ikna etme adına uygulamalarda bulunan XSS zafiyetlerinden yararlanır. Geleneksel phishing saldırıları kurbanlarına çoğunlukla e-posta yolu ile ulaşan, kurbanlarının hesap, kredi kartı vb. bilgileri çalmayı hedefleyen ve kandırmaya dayalı bir yöntemdir. Örneğin, bir kontör satış sitesinde insanlar kontor almak istediklerinde kredi kartı bilgilerini girerler ve herhangi bir satış yapmadan kurbanların kredi kartı bilgileri alınır. Bununla birlikte, rastgele e-postalar göndererek banka sitelerini taklit edip kurbanların kullanıcı adı ve parola bilgilerinin alınması hedeflenebilir. Banka örnekleri, taklit edilen bankanın bire bir kopyasının oluşturulması ile mümkün olmaktadır. XSS yardımıyla oltalama saldırılarında, uygulamada bulunan XSS zafiyeti yardımı ile uygulama içerisinde frame veya pencereler açarak kullanıcının hassas bilgileri alınmaya çalışılır. Sıklıkla karşılaşılan oltalama örneği için genel senaryo şu şekildedir:

“www.ornekadres.com/sinif.aspx?mesaj=<script>alert(güncel);</script>” Satırı çalıştırıldığında mesaj kutusu açılıyorsa XSS zafiyetinden bahsedilebilmektedir. ornekadres.com sunucusu üzerinde çalışacak web sunucusuna sifreguncelle.html dosyası eklenir. Parola değiştirme formunun bir pencere içerisinde açılması için şu şekilde bir kod hazırlanır. “<script>window.open(‘http://www.ornekadres.com/sifreguncelle.html’, ‘pwindow’,width=320, height=150, location=no);” Kod parçası kullanılarak oltalama için gerekli olan adres hazırlanır. Parola değiştirme ekranına kullanıcı adı veya e-posta adresi gibi ek parametre eklenir. Erişim kayıtlarında kullanıcının girmiş olduğu şifre görülebilmektedir. Kurban uygulamadan çıkış yapmış ve yeniden kullanıcı adı, şifre bilgisinin istendiği bir form ile de yapılabilir. Bu durumda hedef kurbanın belirli olmasına gerek yoktur. Rastgele kurbanlara e-posta yolu ile zararlı kod parçacığı barındıran adresler gönderilebilmektedir. Iframe içerisinde sahte uygulama açılacaktır. Hazırlanan sahte çerçeveler uygulamada tarayıcının yükseklik ve genişliklerine göre tüm ekranı kaplayacak şekilde ayarlanmaktadır,

bu sayede kullanıcı sahte sayfada olup olmadığını anlayamamaktadır (Demir, 2013).

3.2.3.3. Oturum kimliği açığı

Web uygulamalarının kimlik doğrulaması adımından geçen kullanıcılar oturumlarını saklamak için çerez bilgisini kullanırlar. Yaratılan bir çerezin içeriğinde kullanıcıya ait oturum kimliği de bulunur. Bu oturum kimlikleri, kullanıcı başarılı bir kimlik doğrulama adımını geçtikten sonra özel olarak hazırlanmış tahmin edilmesi zor oturum kodlarından oluşturulurlar ve hem sunucu tarafında hem de kullanıcı tarafında saklanırlar. Oturum kodu değeri başarılı bir oturum açma işlemi sonrasında da kullanılmaya devam ederse, oturum açılmadan önceki çerezi çalan bir saldırgan, kullanıcı oturum açtıktan sonra da bu çerezi kullanarak sisteme yetkisiz erişim sağlayabilir. Kullanıcı giriş sayfalarında kimlik doğrulaması adımı geçilmeden yaratılan çerez bilgisinin kimlik doğrulaması adımından sonra da kullanılmaya devam ettiği durumlarda bu açığın varlığından söz edilebilmektedir (Muharremoğlu, 2012).

Örneğin saldırgan önce web uygulamasına istek yapar ve oturum bilgisi alır. Oturum bilgisi herhangi bir uygulama adresinin sonuna eklenerek kurbanı gönderilir. Kurban da bu uygulamaya kimlik bilgilerini doğrulayarak girdiği anda saldırgan almış olduğu cookie bilgisiyle kimlik doğrulamayı aşarak uygulamaya girebilir (Bilgi Güvenliği Akademisi, 2014).

3.2.3.4. Siteler arası istek sahteciliği

CSRF, uygulama güvenlikçileri tarafından yaygın görülmesinden ve durumun ciddiyetine tam varılmamış olunmasından dolayı uyuyan dev olarak adlandırılmaktadır. CSRF, uygulamaya gelen her isteğin, gerçekten kendi uygulaması üzerinden geldiğinin farkına varılamamasından veya bu kontrolün yapılmamasından kaynaklanmaktadır. Günümüz uygulamalarında sıklıkla karşılaşılan bir durumdur. Önemi, uygulama yazılırken fark edilememekte, gelen her isteğin zaten uygulamaya giriş yapan geçerli kullanıcılardan geleceğinin ön

kabulünden kaynaklanmaktadır. Bir uygulamaya girildiğinde, kullanıcı adı ve parola bilgileri ilgili forma eklenir ve gönderildiğinde ilgili uygulama kullanıcıyı tanır, yapılan isteklere cevap vermeye başlar. İstekler istemciden, ilgili uygulamaya doğru gönderilmektedir. İlgili uygulama da yapılan her isteğe tarayıcılarda bulunan çerez bilgileri ve ilgili uygulama kullanıcıyı çerez bilgileriyle tanır. HTTP protokolünün yapısı gereği tarayıcı, giden isteğin gerçekten uygulamayı açan pencereden mi yoksa farklı bir pencereden mi gittiğini genel itibari ile kontrol etmez. Dolayısıyla kullanıcı herhangi bir araç yardımıyla istek yapıyor olsa bile, ilgili tarayıcının kapsamı içerisindeyse yapılan isteğe çerez her zaman atanacaktır. Yani kurbanın bilgisi olmadan, ziyaret etmekte olduğu siteye kurbandan geliyormuş gibi yine kurbanın girdiği başka bir siteden isteklerin yollanması ile saldırı gerçekleşir (Demir, 2013). Örneğin, bir kullanıcı, hesabıyla oturum açıp alışveriş yaptığı sitede alışveriş sırasında sepete ekleme yapılabilsin ve sepettekiler silinebilsin. Sepettekilerin silinebilmesi için istemciden istek geldiği düşünölsün. GET isteği ile sunucuya, kullanıcı alışveriş sepetindekileri silmesini isteyen parametre gönderilir. Sunucu tarafında oturum kontrolü yapılır ve geçerli bir oturumdan geldiği için istek yerine getirilir. Bir saldırgan burada bulunan açıklıktan faydalanmak için bir web sitesi hazırlar. Hazırlamış olduğu sayfa içerisine açıklığın bulunduğu alışveriş sitesine istek yapmayı sağlayan “” etiketini koyar. Kaynak değeri olarak alışveriş sitesinde kullanıcı sepetini boşaltan istek adresini yerleştirilir. “” etiketi, “width” ve “height” değeri ayarlanarak sayfada göstermeyebilir.

Sonuçta, sayfa açıldığında alışveriş sitesine kullanıcı sepetini sil isteği gönderilmektedir. İstek, geçerli kullanıcı hesabından gelmemişse herhangi bir işlem yapılmayacak, geçerli bir kullanıcı hesabından gelmişse işlem yapılacaktır. Bir kullanıcı alışveriş sitesinde, kendi hesabıyla oturum açmakta ve sitede gezinmeye başlar ardından sepetine bazı şeyler ekler. Bu sırada saldırganın hazırlamış olduğu betik içeren sayfayı açar. Saldırganın hazırlamış olduğu sayfada bulunan etiketi ile sepetindekileri silecek istek alışveriş sitesine gönderir ve kullanıcı sepeti boşalır. Burada kullanıcının oturum bilgisi isteğin oturum bilgisi olarak isteğe eklendiğinden yetkilendirme ile ilgili bir problem ortaya çıkmayacaktır ve silme işlemi gerçekleşip işlem devam edebilmektedir.

Kullanıcının bu gönderilen istekten haberi yoktur. Burada saldırgan ciddi sonuçlar elde edebilecek başka betikler de hazırlayabilecektir. Bunların arasında, New York Times (nytimes.com), ING Direct, Metafilter ve YouTube gibi çok kullanılan siteler mevcuttur. Fakat şu anda bu sitelerdeki açıklık kapatılmış durumdadır (Zeller ve Felten, 2008). ING Direct bankasında bulunan zaafiyet ile kullanıcı hesabından saldırgan hesabına para transferi yapılabilirdi. Aynı şekilde saldırgan YouTube kullanıcısının favori videolarına istediği videoları ekleyebilmekte hatta kullanıcının sadece ailesinin ve arkadaşlarının izlemesine izin verdiği videolara erişebilmekteydi (Çeri, 2009).

XSS ve CSRF birbiri ile ilişkili açıklardır ve uygulayı hangilerinin etkilediğini belirlemek zordur. Eğer bir uygulama XSS den etkileniyorsa, aynı zamanda CSRF'den de etkileniyordur. Çünkü CSRF bir XSS açığını kullanabilir. Fakat, CSRF'ten etkileniyor olan bir uygulama XSS'den etkilenmiyor olabilir. Bu iki saldırı şeklindeki ana fark saldırının nasıl yapıldığıdır. XSS giriş doğrulamayı atlatma ile ve bir sayfaya direkt olarak kod enjekte ederek yapılır. CSRF ise orada olan açıklığın kötüye kullanılarak yapılmasıdır.

Saldırganlar, CSRF saldırıları için hedef olarak çoğunlukla uygulamalar da oturum açmış kullanıcıları seçer. Çünkü oturum sahibi olmayan kullanıcının sıradan ziyaretçi olacağı ve ziyaretçinin yapacağı işlemi, saldırganın da yapabilmesi olasıdır. Fakat uygulama geliştiricilerin, oturum açan kullanıcıların yaptığı işlemleri geçerli sayması belki de bu açıklığın en büyük sebeplerinden biridir. Yazma, değiştirme ve silme gibi tüm operasyonların GET metodu yerine POST metoduyla yapılıyor olması gerekmektedir. Yapılan her isteğe karşın http Referer başlığı kontrol edilebilir. Dolayısıyla uygulama geliştirici Referer başlığına bakarak, yapılan isteğin kendi uygulaması üzerinden gelip/gelmediğini doğrulamış olur. Önceki çalışmalarda Flash ve XMLHttpRequest ile bu başlığın manipüle edilebildiği görülmüştür. Referer kontrolü tamamen yeterli bir çözüm değildir. CSRF önleyici token değerinin üretilip oturumlar arası post işleminden sonra bu değerın eşleştirilmesi problemi büyük ölçüde ortadan kaldıracaktır. Karakter doğrulama uygulamasının kullanılması, adres yeniden yazma (url rewrite) tekniği ile karmaşık adreslerin belirlenmesi, java ve. net projeleri için

CSRFGuard eklentilerinin kullanılması CSRF zafiyetlerini büyük ölçüde engellemektedir (Demir, 2013). Açıklıktan korunmak ve nesnelere erişim için kullanılan kod, dosya adı gibi değerlerin değiştirilmesinin imkânsızlaştırılması için maskeleyme yöntemleri kullanılarak anlamsız değerler ile taşınması gerekmektedir.

Örneğin “http://www.bgabank.com/?sayfa=profil.php&musteriID=35” yerine “http://www.bgabank.com/?sayfa=profil.php&musteriID=14ab134dad3422dadcedadada12a56” gibi maskelenmiş bir kod (ID) değeri kullanılarak sorgu gerçekleştirilmelidir. Buradan alınan değer belli bir fonksiyona tabi tutularak çözülüp, elde edilen bu değer veritabanından sorgulanabilir (Erdik, 2014)

3.2.3.5. Dizin gezinimi

Dizin gezinimi saldırıları; path traversal, directory traversal, dot-dot-slash, directory climbing olarak da bilinmektedir. Bu zafiyetin asıl kaynağı; uygulamaların dosya veya izin erişim yöntemlerindeki hatalardan oluşmaktadır. Dosya erişimindeki yöntem hataları genel itibari ile uygulama kullanıcılarından alınan girdilerin gerekli kontrollerden geçirilmeden dosya erişim işlevlerinde kullanılmasından kaynaklanmaktadır. Dizin gezinimi saldırısının amacı, uygulamanın kullanmış olduğu veri tabanı kullanıcı adı, parolası, işletim sistemi kullanıcı ve parola bilgileri, uygulama kaynak kodu, yapılandırma dosyaları vb. hassas bilgi içeren dosyalarına erişim sağlamak, uygulama ve işletim sistemi üzerinde önemli bilgiler elde edebilmektir. Günümüz uygulama sunucularının çoğu yetkili bir kullanıcı ile çalışsa da hala çok sayıda yetkisiz kullanıcı ile çalışan yapılar da bulunmaktadır. Bu sebeple izin gezinimi saldırıları uygulamaya yönelik ve işletim sistemine yönelik saldırılar olarak ikiye ayrılır.

Dizin gezinimi zafiyetinin tespiti için öncelikle dosya veya izin erişimlerinde kullanılan girdi noktalarının keşfi yapılmalıdır. Uygulama üzerinde bulunan her girdi noktasında zafiyetin varlığından söz etmek mümkün olmayacağı gibi her noktada bunun denenmesinin de bir anlamı bulunmamaktadır. Şüpheli

noktaların bulunabilmesi için adres satırındaki view, page, path, template, url, item gibi parametre isimleri veya .txt ve .doc vb uzantılı dosyaları çağıran girdi noktaları dizin gezinimi zafiyeti barındıran şüpheli noktalar olarak tespit edilebilir (Demir, 2013).

Dizin gezinimi saldırılarına karşı alınabilecek en başarılı yöntem; uygulamanın önceden erişeceği dizin ve dosyaların yine uygulamanın erişeceği bir yerde bulunduruyor olması ve erişim esnasında buradan kontrol edilerek doğrulanmasıdır. Eğer uygulamanın çağıracağı dosya ve dizinler bir dizide tutuluyorsa gelen her parametreyi bu hash içerisinde bulup, ardından karşısındaki sayfa çağrılabilir. Ayrıca bir dizinin elemanı olup olmadığını kontrol ederek arkasına uzantı ekleyip, çalıştırılabilir. Dinamik dosya isimlerine erişim sağlayan uygulamalarda uygulanması gereken yöntem şu şekildedir;

- Kullanıcıdan alınan dosya ismi girdisinin adres çözme işleminden geçirilmesi gerekir.
- Alınan parametrenin maksimum ve minimum uzunluk kontrolünün yapılması gerekir.
- Dosya uzantısının kontrolünün yapılması gerekir.
- Sadece tek bir “.” Karakterinin bekleniyor olmasını doğrulamak gerekir.
- Dosya isimlerinde sadece harf, sayı ve “_” karakterinin kullanıldığı regex ile kontrol edilmelidir. Fakat bu kontrol “isMatch()” gibi metotlar ile yapılmalıdır.
- Dosya erişimi sağlanmadan önce ana dizin elde edilmelidir. Elde edilen dosya yolunun erişim sağlanması gereken yol olup olmadığı doğrulanmalıdır.

3.2.3.6. Ldap enjeksiyonu

LDAP; Dizin hizmetlerinin sorgulama, erişim, değiştirme gibi görevleri yerine getiren uygulama katmanı protokolüdür. Açık sistem ara bağlantı (OSI) referans modelini kullanan X.500 protokolünün hafifletilmiş halidir. LDAP, dört katmanlı olan TCP/IP modelini kullanır. Dizin, bilgisayar ağlarında bulunan donanımlar, programlar, kişiler vs hakkında bilgiler tutan veritabanlarıdır. Normal veritabanlarına göre veriler istenilen yapıda tutulabilirken, Dizin’ de bu düzen hiyerarşiktir. Hiyerarşik olması da okuma, yazma kısaca erişim anlarında performanslı olması demektir. Ayrıca, veritabanlarındaki gibi “Rollback” ve “Transaction” gibi işlemler desteklenmemektedir. Dizin’ de istenilen amaç,

sadece veriye ulaşmak olduğundan veritabanlarındaki gibi bilgi girişinde karmaşık algoritmalar kullanılmaz.

LDAP sistemindeki başlıca dizin servisleri şunlardır:

- AD, Microsoft'un Windows ağları için geliştirdiği dizin servsidir.
- NIS, Sun Microsystems'ın geliştirdiği, "Yellow Pages" olarak da bilinen dizin servsidir.
- Open Directory, Apple'ın MacOS X server'ları için geliştirdiği dizin servsidir.
- OID, Oracle'ın kullandığı dizin servsidir.
- DNS, İnternet için kullanılan IP-isim rehberi dizinidir (Kırbaş, 2014).

LDAP enjeksiyonu, kullanıcı girişine dayalı LDAP cümleleri oluşturarak web uygulamalarını sözmürmek için kullanılır. Bir uygulama kullanıcı verilerini düzgün girmede başarısız olursa, yerel proxy kullanılarak LDAP sorguları değiştirilebilir. Yetkisiz sorgulara izin vermek gibi keyfi komutların yürütülmesine ve LDAP ağacındaki içeriklerin modifiye edilmesine neden olabilir. SQLI' da bulunan benzer sözmürü teknikleri LDAP enjeksiyonunda da kullanılabilir (Owasp, 2009). Aşağıdaki kod satırı LDAP enjeksiyonuna ait örneği göstermektedir.

```
<input type="text" size=20 name="userName">Insert the username</input>  
String ldapSearchQuery = "(cn=" + $userName + ")";  
System.out.println(ldapSearchQuery);
```

LDAP enjeksiyonu, kullanıcı girdilerinden LDAP cümleleri oluşturan web sitelerini sözmürmeye yarayan bir saldırı tekniğidir. LDAP protokolü, internet taşıma protokollerinin üzerinde çalışır. Web uygulamaları, dinamik web taleplerini karşılamak için, kullanıcı girdileri ile oluşturulan LDAP cümleleri kurar ve kullanır.

Eğer kullanılan bu kullanıcı girdileri düzgün bir şekilde denetlenmezse, bir saldırgan LDAP cümlesinin oluşumunu değiştirebilir. Saldırgan LDAP cümlelerini değiştirebilirse, bu işlem komutu çalıştıran ögenin haklarıyla çalışır. Bu haklar LDAP ağacındaki her şeyin sorgulanmasını, değiştirilmesini veya silinmesini sağladığı için, ciddi güvenlik problemlerine sebep olabilir.

```
userName = Request.QueryString("user")
```

```
if( userName = "" ) then  
Response.Write("<b>Geçersiz talep. Lütfen geçerli bir kullanıcı ismi ve parolası giriniz</b><br>")  
filter = "(uid=" + CStr(userName) + ")" ' kullanıcı girdisini arıyor  
'LDAP objesini oluşturuyor ve temel dn'i tanımlıyor  
Set ldapObj = Server.CreateObject("IPWorksASP.LDAP")  
ldapObj.ServerName = LDAP_SERVER  
ldapObj.DN = "ou=people,dc=spilab,dc=com"  
'arama filtresini tanımlıyor  
ldapObj.SearchFilter = filter  
ldapObj.Search  
While ldapObj.NextResult = 1
```

Saldırgan yukarıdaki satırlarda LDAP sunucusu üzerindeki sorguda mutlak kontrole sahiptir ve sorgusunun sonuçlarını, bütün sonuçlar ve özellikler kullanıcıya tekrar gösterildiğinde alacaktır. Filter parametresi LDAP sözcük diziminin değiştirilerek sistemin aldatılması amacını taşır (Kırbaş, 2014).

3.2.3.7. Javascript enjeksiyonu

JavaScript, web sayfalarında dinamik içerik sağlamak ya da kullanıcıyla iletişim kurmak amacıyla kullanılan, istemci taraflı yorumlanan bir dildir (Wikipedia, 2015). HTML, tam olarak bir programlama dili olarak ifade edilemeyeceğinden, HTML kodlarıyla yapabilecekler bir web uygulamasının genellikle görsellik kısmıyla ilgili olduğundan ve saldırının ana teması HTML kodlarının enjekte edilmesi olduğundan dolayı en basit düzeyde ve zararı en az olan saldırı olarak değerlendirilmektedir. Bu nedenden dolayı, JavaScript komutları da HTML etiketleri gibi tarayıcılar aracılığıyla yorumlandığından ve JavaScript komutları ile yapabilecekler, JavaScript Enjeksiyon tekniğini doğurmuştur (Yılmaz, 2009).

Javascript, yararlı uygulamalar ve fonksiyonların her birinde kullanılabilir. Ama bununla birlikte test edilmesi gereken bazı güvenlik sorunları ortaya çıkmaktadır. Javascript sadece iyi amaçlar için değil aynı zamanda kötü niyetli amaçlar içinde kullanılabilir. Javascript kullanılarak saldırgan form içerisindeki var olan bilgileri, tarayıcıdaki cookie bilgilerini ve web sitesindeki diğer değerleri değiştirebilir. Gerçekleştirilmek istenen parametre manipülasyonu her türü Javascript enjeksiyonu ile yapılır. Herhangi bir oturumda javascript çalıştırmak için kullanıcı, tarayıcının adres çubuğuna - http:// gibi özel bir javascript kodu girmelidir. Javascript kullanarak kullanıcılar

mevcut cookie ayarlarını deęiřtirebilir. Bu bazı temel javascript fonksiyonları ile gerekleřtirilebilir. Mevcut cookie ierięini grmek iin, ařaęıdaki javascript kodlarını yeterlidir. Adres ubuęuna “javascript:alert(document.cookie);” adresi kopyalandıęında cookie bilgilerini listeleleyen bir pencere aılacaktır. Kt niyetli bir kullanıcı cookie deęerlerini deęiřtirebilmek iin javascript kullanabilir.

Devre dıřı bırakılan, gizli formlar ieren html formlarındaki deęerler javascript kullanarak deęiřtirebilmektedir. “javascript:void(document.forms[0].email.value = “test@test.com”);” satırı email giriř verisinin deęiřtięini gstermektedir. Hangi parametrenin deęiřmesi gerektięini belirlemek ve nasıl deęiřtięini bilmek iin html sayfanın kaynak kodu grntlenmek zorundadır.

Javascript Enjeksiyonundan korunmak iin giriřler beyaz listeye gre doęrulanmalıdır. Eęer kara liste kullanılırsa yksek satır sayısı gibi kodlama sorunlarıyla karřı karřıya kalınabilir. Javascript kullanarak doęrulama kullanılmamalıdır. Javascript kullanarak html sayfayı deęiřtirmek ok kolaydır. Javascripti sadece parametre ve cookie leri deęiřtirmek iin deęil, aynı zamanda javascripti dinamik sayfalarda farklı bir řekilde alıřtırmak iin veya zararlı řeyler iin enjekte edilir. Proxy programları kullanarak parametre hileleri yapılabilmektedir (Akgn vd, 2005).

3.2.3.8. Xml sorgulama dili enjeksiyonu

XML sorgulama dili (XPATH), xml dokmanları iinde sorgu yapmak iin kullanılan ve yorumlanan bir dildir. Web uygulamaları genellikle veri saklamak iin XML dosyalarını kullanmasa da ayarlama verisi saklama amacıyla bu dosyaları kullanılabilmektedirler. XPATH enjeksiyonu prensip olarak SQL enjeksiyonuna benzese de veri elde edebilmek iin XML veri deposu eleman isimlerinin bilinmesi gerekmektedir. SQLI’ da olduęu gibi farklı kořullara verilen tepki farklılıklarından yola ıkarak ve XPATH dilinin “substring” fonksiyonundan faydalanarak XML elemanlarının ierikleri karakter karakter elde edilebilir. SQLI aıklıęı bulunduęundan řphe edilen ancak cmlecięin bilinemedięi durumlarda XPATH enjeksiyon ihtimali dřnlmelidir.

3.2.3.9. Kod enjeksiyonu

Genel tabiri ile kod enjeksiyonu, saldırganın hedef web uygulaması üzerine zararlı kod veya kod parçacığı eklemesidir. Kod enjeksiyonu, kod bütünüünün veya parçalarının uygulamaya başarılı bir şekilde gönderilmesi ve daha sonra uygulama tarafından sanki kendi proje parçasıymış gibi çalıştırılmasıdır. En popüler kod enjeksiyonu PHP teknolojisinde bulunan ve birçok sunucunun saldırganlar tarafından ele geçirilmesini sağlayan uzak dosya (RFI) ve yerel dosya (LFI) zafiyetleridir. Büyük çoğunlukla PHP tabanlı uygulamalarda bulunsa da Java ve ASP.NET uygulamalarında da görmek mümkündür. Karmaşık kodları daha iyi yönetebilmek için modüleriteyi arttırmak amacı ile PHP çatısında, bir PHP dosyasına başka bir PHP dosyasının dahil edilmesi özelliği vardır. Bu işlem için “include(), require(), include_once(), require_once()” fonksiyonları kullanılır. PHP “include/require” fonksiyonlarına parametre olarak verilen değerler saldırganlar tarafından değiştirilebilirse, sistem üzerinde PHP’nin erişebileceği diğer sistem ve uygulama dosyaları da çalıştırılacak PHP dosyaları içine kopyalanıp, saldırgana cevap olarak dönecektir. Örneğin, Php include fonksiyonu için, kullanıcı kullandığı PHP web uygulamasının dil değiştirme özelliğini kullanmak istediğinde yapması gereken TR içerik üretecek linke tıklamasıdır. İşlem sonucunda index.php dosyası içerisinde bulunan include fonksiyonu, istenen parametre değerinin sonuna .php uzantısı ekleyerek, kök dizin içinde tr.php isminde bir dosyayı index.php içine yerleştirecektir. Daha sonra bu hali ile php dosyası çalıştırılacak ve içerik kullanıcının tarayıcısına gönderilecektir.

Saldırgan “lang” parametresinin değeri olarak “/etc/passwd%00” atayıp, sunucuya aynı şekilde gönderdiğinde, index.php içindeki include fonksiyonu lang değeri olan “/etc/passwd%00” string’inin sonuna .php uzantısını ekleyerek “/etc/passwd%00.php” dosya ismini elde edecektir. Bu dosyayı sistem üzerinde arayan PHP çatısı string %00 ile bittiğinden sistemdeki kullanıcılar ile ilgili önemli bilgiler barındıran /etc/passwd dosyasının içeriğini index.php dosyasına yapıştıracak ve çalıştırıp, saldırgana gönderecektir. PHP, C programlama dili ile yazılmış olduğundan, bir karakter dizisi “%00” yani “\0” null karakteri

barındırıyorsa, bu string'in bittiğini işaret edecektir. Bu şekilde “/etc/passwd%00.php” değeri C dosya işleme fonksiyonları için “/etc/passwd” demektir. Bu saldırı çeşidine LFI denmektedir.

Saldırganın “/index.php?lang=/http://www.saldirgan.com/Webshell.txt?” şeklinde yapacağı saldırıda PHP çatısı allow_url_fopen php.ini direktifi enable şeklinde yapılandırılmış ise, index.php içindeki include fonksiyonu saldırırganın gönderdiği değer http ile başladığından bir http isteği ile saldırırgan.com’dan webshell.txt?.php parametresini isteyecektir. Soru işareti http url parametrelerini belirten ayraç olarak algılanacağından, webshell.txt dosyası olduğu gibi index.php içine www.saldirgan.com dönünce yapıştırılacaktır. Bu dosya saldırırganın çalıştırmak istediği işletim sistemi komutlarını alıp çalıştırabilen bir PHP kod kümesi barındırdığından, artık saldırırgan index.php’ye lang parametresinin yanında çalıştırmak istediği komutları da gönderebilecektir. Örneğin;”/index.php?cmd=netstat-tpln&lang=http://www.saldirgan .com/webshell.txt?” RFI saldırısı için saldırırganın ayrıca bir web sunucusu kurması ve bu sunucunun kök dizinine PHP kodunu yerleştirmesi gerekir. LFI içinse, saldırırganın farklı şekillerde web sunucusu upload edebileceği webshell dosyaları yeterli olabilmektedir. Php uygulamalarında LFI ve RFI zayıflıklarını bulmak için Fimap uygulaması kullanılabilmektedir (Bilgi Güvenliği Akademisi, 2014).

3.2.3.10. Bellek taşması

C programlama dilinde bellek dizi olarak geçer. Diğer bütün veri türleri gibi diziler de sabit ya da değişken olarak sınıflandırılabilirler. Sabit değişkenler, program hafızaya yüklenirken, programın veri kısmına yerleştirilirler. Değişkenler ise program çalışırken, dinamik olarak hafızada program için hazırlanmış özel bölümde yaratılıp, yok edilirler. Ara bellek taşması, bu dinamik değişkenlerin taşıyabilecekleri veri miktarından fazlasını yükleyerek değişkenin sınırlarını aşması olayıdır. Örneğin, 10 byte veri taşıyabilecek bir diziye 20 byte veri kopyalamak, belleğin taşmasına neden olmaktadır.

```
main(){  
char array[10];
```

```
Strcpy (array,"bu mesaj ara bellek taşmasına yol açar");  
return 0;  
}
```

Programın çalışması sırasında ayrılan hafıza boyutu kadar veri işlem görür, diğerleri ise yasak bölge diye nitelenen yığın bölgesine yerleşmek zorunda kalır. Bu işlem bazı durumlarda programın çökmesine ya da çalışmanın durdurulmasına neden olur ve “geçersiz işlem yürütüldü” uyarısı verilmesine neden olarak önemli ölçüde açık oluşturur. Bazı programlarda ise herhangi bir değişim olmaksızın çalışmaya devam edilir. Yapılan bu işlemlerde en önemlisi de, yığın denenen hafıza alanına taşan veri kötü amaçlar için yapılandırılırsa bu durumda saldırganın sistem üzerinde istediği işlemi yapması çok kolaylaşır (Akgün vd, 2005). Arabellek taşma kusurları, web uygulamasının kendisinde veya sitenin statik ve dinamik görünüşünü sunan web veya uygulama sunucusu ürünlerinde bulunabilirler. Yaygın olarak kullanılan sunucu ürünlerinde bulunan arabellek taşmaları, daha fazla kişi tarafından bilinmeye yatkın olup, bu ürünleri kullanan kullanıcılar için kayda değer bir risk oluştururlar. Web uygulamaları, resim üretmek için kullanılan grafik kütüphaneleri gibi kütüphaneler kullandığında, kendilerini potansiyel arabellek taşmalarına açık bırakırlar.

Programlama dillerinde kullanılan kütüphane dosyalarının doğası gereği güvenliği zayıftır ve saldırganlara açık bir hedef teşkil ederler. Saldırgan tarafından kütüphane dosyasında bulunan herhangi bir zayıflık, aynı kütüphane dosyasını kullanan diğer programlarda da mevcut olacağından, bu tür açıklar saldırganların iştahını kabartan bir saldırı aracı haline dönüşmektedir. Kütüphane dosyalarını kullanmak ara bellek taşmalarına neden olabilmektedir. Örneğin, veritabanı problemlerine yol açabilecek, muhtemel tehlike oluşturabilecek HTML kod ve karakterlerinin filtrelenmesi gerekir. Kesme işareti, tırnak işareti ve & işareti ayrılmış işaretler kullanıcı girişlerinde bulunmamalıdır, aksi halde uygulama çökebilir. Bu tür karakterlerin süzme işleminden geçirilerek filtrelenmesi ara bellek taşıma problemlerinin önüne geçecektir. Güvenli Kodlama için kullanıma geçmeden önce bütün yazılımların güvenlik testlerinden geçmesi gerekir. Bir saldırgan gibi hackleme denemeleri yapılmalı, sorunlar varsa giderilmelidir. Eğer bu denemeler sırasında yazılım çöküyorsa, saldırganların

kullanabileceği bir problem olduğu açıktır ve düzeltilmesi gerekmektedir (Kocaoğlu, 2010).

3.2.3.11. Dos saldırıları

DOS, hizmet aksatma amaçlı bir saldırı çeşididir. Bir sisteme yapılan düzenli saldırılar sonucunda sistem çalışamaz ve hizmet veremez hale gelir. Ayrıca DOS saldırılarıyla hedef sisteme ait kaynakların tüketilmesi de amaçlanır. Bu saldırı önemli sunucuların servis vermeyi durdurması gibi büyük sorunlara yol açabilir. DOS saldırıları 2 aşamada gerçekleşmektedir. Toplu olarak güvenliği kırma aşamasında, DOS saldırısı yapacak olan sistemlere ulaşılır ve saldırıyı gerçekleştirecek olan programlar yüklenir. Bu sistemler, ilk zarar görecektir. DOS saldırıları aşamasında hedef sitelere saldırı yapılır ve bunun için de ilk aşamada saldırıyı gerçekleştirecek programların yüklendiği bilgisayarlar kullanılarak hedefe saldırı gerçekleştirilir (Seyir Defteri, 2013).

Bilgisayar korsanı, hiçbir şeyden habersiz olan bilgisayar kullanıcılarına bir program yüklemekte, bu şekilde binlerce kullanıcıyı izleyerek sistemlerine zarar verebilmektedir.

Bilgisayar korsanının belirlediği bir güne kadar bu programlar sessiz sedasız bir kenarda beklemektedir. Bu programların yüklendiği makineler zombi makineler denmiştir. Bütün bilgisayarlar aynı anda, önceden belirlenmiş bir web sitesine, giriş talebi göndermeye başlamaktadır. Bu tür talep gönderen bilgisayar sayısı on binleri bulduğunda, doğal olarak karşı tarafın sunucusu yanıt veremez duruma gelir. Sonuçta da web sitesi çöker, işlem yapamaz ve site sahipleri maddi zarara uğrar. Saldırıda kullanılan programcıkların yüklendiği bilgisayarlar ve kurumsal ağlar, ciddi bir bant genişliği darboğazı yaşamaktadır, çünkü bu saldırılar, büyük miktarda veri gönderimi yapılmasını gerektirmektedir. Saldırıya karşı korunma yolu, bilgisayar ve ağ güvenliği sağlayan çözümlerin kurulmasından ve doğru şekilde ayar yapılmasından geçmektedir (Netcom, 2014). DOS saldırılarının başlıcaları şu şekildedir;

Arabellek Aşımı Saldırıları: DOS saldırılarının en yaygın çeşitidir. Arabellek; hafızada ard arda dizili türdeş veri depolayan hafıza bloğudur. Arabellek aşımı saldırıları; bir internet sitesine, sitenin programcısının gelmesinin beklediği ve bu amaçla veri akışı için planladığı arabellek değerinin karşılayamayacağı kadar çok trafik yollanması sonucu oluşur.

Hizmet Engelleme Saldırıları: Bu saldırı türünde saldırgan, internet üzerinde kullanılmayan IP adreslerini kullanarak birçok hizmet engelleme (SYN) paketini hedef makineye yollar. Hedef makine, alınan her SYN paketi için kaynak ayırır ve bir onay paketini(SYN-ACK), SYN paketinin geldiği IP adresine yollar. Hedef makine, kullanılmayan IP adresinden yanıt alamayacağı için onay paketini defalarca tekrarlar. Saldırgan bu yöntemi üst üste uyguladığında hedef makine ayırdığı kaynaklardan ötürü yeni bir bağlantıyı kaldıramaz duruma gelir ve bu sebepten makineye bağlanılamaz.

Paket Saldırıları: Bir bilgisayara internet üzerinden gelen paketler, bilgisayarda bölünerek aktarılır. Paket verilere ayrıştırılırken, pakette bulunan ofsetler kullanılır. Bu ofset bilgilerinin çakışmaması gerekmektedir. Teardrop saldırılarında, paketi gönderen saldırgan, pakete üst üste gelecek ofsetler ekler. Paketi alan bilgisayar, böyle bir durumu kontrol edebilecek mekanizmaya sahip değilse, sistem çöker.

Aldatma (Smurf) Saldırıları: Bu saldırı türünde, saldırgan hedef bilgisayardan ping isteğinde bulunur. Ancak ping paketi, hedef makinenin IP'sinden geliyormuş gibi görünecek şekilde hazırlanmıştır. Bu durumda ağ üzerindeki bütün makineler, hedef makineye ping atar. Hedef makine bu trafiği karşılayamaz ve bağlantı kesilir.

Servislere Aşırı Yüklenme: Bu saldırı tipi belirli kullanıcı ve servisleri düşürmek için kullanılır. Saldırı yapan kişi özel port ve kullanıcıya birçok İnternet kontrol mesaj iletişim kural paketi gönderir. Bu olay ağ izleyicisi ile kolayca anlaşılır.

Paket Yollama: Servislere aşırı yüklenme' den farklı olarak sistemin normal çalışmasını engellemez. Yine aynı şekilde paketler gönderilir ancak bu defa kullanıcı ya da servisler bu paket gönderimini normal olarak algılar.

3.2.3.12. Ddos saldırıları

DOS, çevrimiçi bir uygulama veya hizmeti sekteye uğratmak amacıyla bir ana bilgisayardan başka bir bilgisayara trafik gönderildiğinde gerçekleşir. DDOS saldırısı, düzenlenen bir saldırıyı daha da güçlendirmek için birden fazla ana bilgisayar kullanıldığında gerçekleşir. Saldırganlar hizmet reddi saldırısına uygun koşulları genelde sunucunun bant genişliğini tüketerek veya sunucunun kendisine zarar vererek oluştururlar. Web sunucuları, Alan Adı (DNS) sunucuları, uygulama sunucuları, yönlendiriciler, güvenlik duvarları ve internet bant genişliği gibi noktalar DDOS saldırılarının hedefleri arasında yer almaktadırlar. Kurumlar çevrimiçi ortamda gittikçe daha fazla iş yaptığı için, DDOS saldırıları ciddi mali kayıplara ve verimlilik kaybına yol açabilir. Saldırganın sızdığı ve zombi yazılımlar kurarak bir nevi kölesi haline getirdiği bilgisayarlar topluluğuna Botnet denir. Zombi yazılımı direkt bilgisayara enjekte edilebileceği gibi, bulaştırılan bir trojan ile bu zombi yazılım indirilip kurulabilir (Verisign, 2014).

DOS tek bir bilgisayar veya ağ üzerinden yapılırken DDOS bir çok bilgisayar ve ağ üzerinden yapılır. Haliyle DDOS saldırısı DOS saldırısına göre çok daha etkili olur. DDOS saldırısı yapılırken saldırıyı yapan kişi aslında web sunucusuna bir paket veri gönderip pakete dair cevap bekler. Ancak sunucuların bunu kaldırabilme limitleri vardır. Google gibi firmalarda olduğu gibi inanılmaz büyüklükte bir sunucu ağına sahip olmayan web siteleri fazla yükü kaldıramayıp çökecek ya da inanılmaz yavaşlayacaktır (W.O.D, 2013). DDOS saldırısı ise, koordineli olarak yapılır ve DOS'a göre daha fazla zarar verir. Bununla birlikte saldırı kolay bir şekilde kimliğini belli etmeden gizlenebilir ve saldırıyı tespit edilmesi zorlaşır. Saldırı DOS saldırısı olursa, yani tek bir ip üzerinden saldırı gerçekleşirse firewall'a takılma olabilir. Fakat DDOS saldırısında çok sayıda makine kullanıldığından, ip tespiti güçleşir ve firewall yakalayamayabilir ve log taşması

sonucu güvenlik duvarı devre dışı kalabilmektedir. Bu nedenle DDOS, DOS saldırısına göre daha tehlikeli ve etkilidir.

1999 yılında Minnesota Üniversitesi öğrencileri tarafından ilki gerçekleştirilen DDOS saldırıları, 2000 yılında Trinoonun, CNN, Yahoo, EBay, Datek gibi siteleri hedef almıştır. 2002 Kök DNS sunucularını hedef alan DDOS atağı gerçekleştirilmesi, 2007 yılında Estonya siber saldırıları, 2008 yılında gerçekleştirilen Gürcistan siber saldırısı, 2010 yılında Wikileaks, TİB, BTK, TÜBİTAK kurumlarına yönelik saldırılar ve 2011 yılında Anonymous'un Malezya, Türkiye gibi ülkelere gerçekleştirdiği saldırılar olarak güncel olma özelliğini korumaktadır.

DDOS saldırısında amaç sisteme sızma girişiminde bulunmak değildir. Bilgisayar sistemlerini ve bunlara ulaşım yollarını işlevsiz kılmak, web sitelerinin e-postaların, telefon sistemlerinin, bankacılık sistemlerinin çalışmamasını sağlamak DDOS'un başlıca hedefleridir. Tüm bunlar politik, ticari ve bahis amaçlı sebeplerden dolayı yapılmaktadır.

HTTP üzerinden yapılan DDOS saldırıları için öncelikle yapılması gereken şey web sunucu loglarını incelemek olmalıdır. Bu loglar içerisinde saldırı tespit sistemi üzerinden engellenebilecek HTTP başlığı, referrer veya user-agent gibi sabit bir veri bulunabilir. Gelen istekler değişik çerezler ile geliyorsa bir dakika içerisindeki web sunucu loglarına bakarak en çok istegi yapan IP adresleri daha sisteme ulaşmadan engellenebilir. Farklı kaynaklardan gelen istekleri bu şekilde engellemek mümkün değildir. Saldırı binlerce farklı IP adresinden gelmekte ve bu IP adresleri sürekli değişmektedir.

Güvenlik duvarı üzerinde TCP Bağlantı limiti koyarak ve eşik değerini düşürerek saniyede gelen aşırı HTTP-GET isteklerini web sunucusuna ulaşmadan engellenebilir. Bu genellikle etkili olan ve saldırganın başarısını azaltan bir engelleme mekanizmasıdır. Bu durumda gelen istek HTTP-GET isteği olduğu için güvenlik duvarı ya da saldırı tespit sistemi (IDP) hangi isteğin doğru hangi isteğin yanlış olduğunu anlayamayacak ve bu durumda siteye normal olarak ulaşmak

isteyen kişilerin isteklerinin de engellenme ihtimali söz konusu olacaktır (İlgin, 2008).

3.3. Yazılım Kod Kalitesi

Yazılım endüstrisindeki gelişmeler, yazılım geliştirme sürecini etkileyerek bu süreci proje yönetimi çerçevesinde geliştirilen faaliyetler haline getirmiştir. Günümüzde artık küçük ölçekli yazılımlar bile sadece bir kişinin değil, iyi yönetilen bir yazılım ekibinin birlikte gerçekleştirdiği, binlerce kod satırından meydana gelen projelerdir. Nesne yönelimli tasarım ve geliştirme, günümüz yazılım endüstrisinde önemli bir yere sahiptir. Dolayısıyla, nesne yönelimli programlama yaklaşımının kullanılması ile birlikte söz konusu yazılımların kalitesini ölçmek için başvurulan yöntemler geleneksel yöntemlerden farklı olup, yazılım kalitesini ölçümleyecek teknikler geliştirilmektedir (Calp ve Arıcı, 2011).

Yazılım dünyasında kalite, kavram olarak herkesin hissedebildiği ancak neticede kimsenin tam olarak tanımlayamadığı soyut ve öznel bir olgudur. Bir disiplin içerisinde kalitenin tam olarak tanımlanabilmesi için gelişmiş ölçme araçlarına ve sağlıklı karşılaştırmalar için tanımlı referans noktalarına gerek vardır (Erdemir vd, 2008). Son yıllarda, yazılım kalite metrikleri çok önem kazanmaktadır. Metrikler, yazılımların birçok yönden değerlendirilmesini sağlar. Bu değerlendirmeler, kaliteli bir yazılım geliştirilmesinde yardımcı olur.

3.3.1. Yazılım kod metrikleri

Yazılım metrikleri, yazılımların ölçülebilen özellikleri ve bu özelliklere sayıların karşı düşürülebilmesi yöntemidir. Metriklerin kullanılmasındaki temel amaç yazılımın tekrar kullanılabilirlik, bakım kolaylığı gibi kaliteyi belirleyen değerlerini ölçebilmek, sorunlu birimleri bulabilmek ve yazılımın gelecekteki durumu hakkında öngöründe bulunabilmektir. Yazılım metrikleri yazılımın kalitesini göz önüne sermeyi amaçlar. İncelenen yazılım belirlenen kalite metrikleri doğrultusunda; Değiştirilebilirlik, test edilebilirlik, bakım yapılabilirlik, kararlılık, modüllerin kullanılabilirliği gibi açılardan

değerlendirilmektedir. Bu kalite elemanlarının bazıları ölçülen metriklerle doğrudan formülleştirilip gösterilebilmektedir.

Kod metrikleri, kaynak kodları incelemek ve çeşitli ölçülere göre yazılımı derecelendirmek için kullanılan matematiksel ve istatistiksel yöntemlerdir. Kod metrikleri ile geliştiriciler hangi tiplerin veya metotların yeniden ele alınması gerektiğini, düzenlenmesi gerektiğini anlayabilirler. Kod metrikleri kullanılarak, koddaki potansiyel sorunlar erkenden ortaya çıkartılmış ve problemli alanları yeniden düzenleyebilmek için yeterince zaman kazanılmış olur (Karagedik, 2010). Kod metrikleri ile sadece yazılımın sağlığı incelenmiş olmaz, kod metrikleri aynı zamanda koddaki bakımı mümkün olmayan ve karmaşık noktaların da görülmesini sağlar. Kod metrikleri en genel tanımıyla; kaynak kodları incelemek ve çeşitli ölçülere göre yazılımı derecelendirmek için kullanılan matematiksel ve istatistiksel yöntemler olarak tanımlanabilir. Kod metrikleri; Yazılan sınıflarda, metotlarda, projelerde aksayan ve karmaşılaşmış noktalar varsa bunların bulunmasına yardımcı olur (Kırdınlı, 2010). Metrikler sınıf ve metod kategorisinde incelenebilmektedir. Nesneye yönelik metrikler arasında en çok kullanılan metrikler Chidamber ve Kemerer (CK) metrikleridir (El-Amam, 2002). Bu metrikler aşağıdaki gibi listelenmiştir.

Sınıfın Ağırlıklı Metot Sayısı – Weighted Methods per Class (WMC)
Kalıtım Ağacının Derinliği – Depth of Inheritance Tree (DIT)
Alt Sınıf Sayısı – Number of Children (NOC)
Nesne Sınıfları Arasındaki Bağımlılık – Coupling Between Object Classes (CBO)
Sınıfın Tetiklediği Metot Sayısı – Response for a Class (RFC)
Metotların Uyumluluğu – Lack of Cohesion in Methods (LCOM)

Metot seviyesinde ise Halstead ve Mc Cabe metrikleri incelenebilmektedir. Mc Cabe metrikleri kategorisinde ise istatistiksel metrikler kullanılabilmektedir. Bu metriklerin en önemlileri aşağıdaki gibi listelenmiştir.

Cyclomatic Complexity,
Lines of Code,
Source Line of Code

Statik kod analizi ile üretilen metrikler sayesinde, yazılım kalitesi ölçülebilir ve iyileştirilebilir hale gelmektedir. Kaynak kod kalitesi; Standartlara uygunluk, idame ettirilebilirlik, derleyici uyarılarının azlığı gibi niteliklerle; yazılım ürün kalitesi fonksiyonellik, güvenilirlik, kullanılabilirlik, verimlilik, güvenlik gibi metriklerle ölçülmektedir. Yazılım geliştiriciler ve yöneticiler kod tabanlı metriklerden değişik amaçlar için yararlanmaktadır. Sistem seviyesinde tahminlerde bulunulması, erken ölçümlerle güvenlik açısından sıkıntılı bileşenlerin önceden belirlenmesi ve güvenli tasarım ve programlama yönergelerinin geliştirilmesi bu amaçlara örnek olarak verilebilmektedir (El-Amam, 2002).

Yazılan kodda mümkün olduğunca az hata çıkması için, kodu yazan kişi tarafından hatanın giderilmesi ve diğer bileşenlerin daha az etkilenmesi istenir. Metriklerin yaptığı iş, kodun belirli kriterlere göre değerlendirmesini yapıp geliştiricilere bazı değerler hakkında fikir vermesidir. Geliştirilen projedeki kodların satır sayısı, yorum satırı sayısı, sınıflar, ortalama metot ve içerdiği değişken sayısı gibi birçok metrik kodun çeşitli konulardaki değerlerinin izlenmesini sağlamaktadır. Geliştirme süresince sürekli ölçüm ve düzeltmeler ile, yazılım metrikleri tasarım ve geliştirme sırasında ileride daha büyük maliyetler getirecek pek çok hatayı engellemektedir. Kod metrikleri ölçümleri gerçekleştirilen projelerin türüne göre sınıflara ayrılmaktadır. Kodlara bağlı metrikler şu şekildedir;

- **Kod Kapsamı (Code Coverage)**

Yazılan testlerin kodun ne kadarını kapsadığını ölçer. Code coverage için %80 gibi bir oran oldukça iyi görünse de aslında az sayıda basit test yazarak dahi bu orana ulaşılabildiği gözlemlendiği için hedeflenen oranın %100 olması önerilir.

- **Uyumluluk (Cohesion)**

Bu ölçüm sınıfın sorumlu olduğu işlerin kendi içindeki uyumluluğunu ölçer. Her sınıfın tek bir sorumluluğu olmalıdır. Uyumluluk LCOM adı verilen ölçüt ile bulunur. Değişik türleri bulunan LCOM, sınıfta yer alan alanlara metodların ortak

erişim sayısını temel alan bir ölçümdür. LCOM için bu değer 0 ile 2 arasında değişir ve 1' in üzerindeyse sınıf bölünmelidir.

- **Etkileşim (Coupling)**

Bir nesnenin diğeri ile etkileşime girmesine denir. Program içerisinde mutlaka etkileşim olacaktır, ancak bu ilişkinin nesnelerin implementasyon detaylarından mümkün olduğunca bağımsız olması istenir. Farklı ilişki türleri üzerinden ölçülebilir. En çok kullanılanlardan birisi CBO' dur.

- **Nesneler Arası Etkileşim (Coupling Between Objects- CBO)**

Miras alınan sınıflar hariç, sınıfın çalışmak için ihtiyaç duyduğu sınıf sayısıdır. Kütüphanelerde bu sayı yüksek olabilir ancak çalıştırılabilir sınıflarda 6 ile 10 arası makul kabul edilebilir.

- **Çevrimsel Karmaşıklık (Cyclomatic Complexity-CC)**

Çevrimsel karmaşıklık kodun ne kadar karmaşık olduğu hakkında fikir veren en önemli ölçümlerden biridir. Bunu yapmak için kodun içerisindeki "if,else, while, switch, for" gibi yapıların sayısını kullanır. Kabul edilen eşik değer 10' dur. Bunun yanı sıra zorunlu karmaşıklık denilen bir metrik daha mevcuttur ancak çevrimsel karmaşıklığın daha etkili bir metrik olduğu belirtilmektedir. Çevrimsel yoğunluk, koddaki karar noktalarının toplam çalıştırılabilir koda oranıdır. 0.14 ile 0.42 arasındaki değerler için kodun basit ve anlaşılabilir olduğu kabul edilmektedir.

- **Sınıftaki Method Sayısı (Response For Class-RFC)**

Bir sınıfta yazılan ve çağırılan toplam metodların sayısıdır. Bu değer yükseldikçe kodun bakımı zorlaşır. Önerilen eşik değer 55 dir.

- **Kalıtım Derinliği (Depth of Inheritance Tree-DIT)**

Miras ilişkisinde sınıfın üzerinde kaç tane atası olduğunu gösterir. 6' nın üzerinde ise test edilebilirliğin çok düşük olduğunu, 2 nin altında ise nesne yönelim ilkerinin fazla kullanılmadığına işaret etmektedir. Uygulamanın genelinde 2 ve 3 düzeyinde olması hedeflenmelidir.

- **Aşırı Yüklenmiş Metot Sayısı (Specialization Index)**

Kod karmaşıklığını ve bakım maliyetlerini arttırmamasından dolayı aşırı yüklenmiş fonksiyon sayısının mümkün olduğunca az olması istenmektedir. Bundan dolayı $SIX = (\text{Aşırı Yüklenmiş Metot Sayısı} * DIT) / NOM$ şeklinde hesaplanır. %120' ye kadar normal kabul edilmektedir (Ünsal, 2010).

Geleneksel metrikler ise daha çok kod üzerinde istatistiksel ölçümler yapılarak elde edilmektedir.

- **Kod Büyüklüğü (LOC)**

Yazılımın boyutu satır sayısı ile ölçülür. Satır sayısı ölçümü programın tüm satırlarının sayılmasıdır. Satır büyüklüğü pek çok dile uyarlanabildiği için oldukça kullanışlı ve ölçümü beklenen bir metriktir. Ancak satır büyüklüğü üzerine yorum yapmak için dilin karmaşıklığı, projenin karmaşıklığı gibi faktörler de göz önünde bulundurulmalıdır. Yazılım istatistiklerini biriktiren şirketler yeni projelerin faktörlerini göz önüne alarak, kestirimlerde bulunabilirler. Satır büyüklüğünün küçük olması yazılımlar için beklenen çıktılar arasında olmalıdır.

- **Kaynak Kod Büyüklüğü (SLOC)**

Yazılımların, yorum satırları ve boş satırlardan (Non-comment Non-blank NCNB veya Source Lines Of Code SLOC) arındırılmış halidir.

- **Yorum Oranı (CLOC)**

Yorum oranı, program için hazırlanmış yorum satırlarının (kod ile birlikte ya da dışarıda) , toplam programın yorum ve boşluk içermeyen satır sayısına (SLOC) bölümü ile bulunur. Yüksek yorum oranı programların anlaşılabilirliğini arttıran ve bakımını kolaylaştıran bir faktördür. Yorum oranının %20 ila %30 arası olması tavsiye edilen bir durumdur.

Diğer geleneksel metrikler arasında ise, yorum satırı sayısı (CLOC) ve boş satır sayısı (BLOC) bulunmaktadır.

Sınıf temelli metrikler ise nesne yönelimli yazılımların geliştirildiği ortamlarda kullanılmaktadır. Nesne yönelimli programlamada temel birim sınıftır. Sınıf ve onun temel elemanları üzerinden alınan metrikler tüm ürün kalitesi hakkında genel fikirler verir. Nesne yönelimli metrik kategorisine giren sınıf temelli metriklerin en önemlisi aşağıda verilmiştir.

- **Sınıf Başına Method Sayısı (NOM)**

Sınıf başına düşen metot sayısı, sınıfın karmaşıklığı açısından bir göstergedir. Aynı zamanda fazla metot sayısı arabirimin ayrılması prensibine aykırı durumların da habercisidir. (Ortalama method sayısı = Toplam Method Sayısı / Toplam sınıf sayısı formülü ile ortalama method sayısı elde edilir.) NOM değerinin 20' nin altında kalması tercih edilmelidir. Bu tip metotlar ayıklanmış haldeyken sınıfın metot sayısının 10' u geçmemesi tercih edilmelidir. Büyük NOM değerleri anlaşılması, yeniden kullanımı ve bakımı güç sınıflara işaret eder (Dirik, 2011).

Chidamber ve Kemerer (CK) metrikleri, sınıf temelli metrikler kategorisinde incelenmektedir. CK metrikleri arasında sınıf başına ağırlıklı metot, kalıtım ağacının derinliği, alt sınıf sayısı, nesneler arası eşleşme, sınıf yanıt sayısı, uyumsuzluk bulunmaktadır.

- **Sınıf Başına Ağırlıklı Method (WMC)**

Çevrimsel karmaşıklık değerleri toplamının sınıf sayısına bölümü ile bulunur. WMC değerinin 100' ün altında olması tercih edilmelidir. WMC, sınıf karmaşıklığı hakkında fikir veren en önemli donelerden biridir. Bu metriğin faydaları, sınıf başına ortalama metot sayısı gibi değerlendirilebilir. WMC değerinin 100' ün altında olması kabul edilebilir bir değerdir. WMC değeri NOM' dan farklı olarak sınıfın karmaşıklığı hakkında daha net bir fikir verir. WMC sınıfın karmaşıklığı hakkında fikir verir ve harcanacak geliştirme ve bakım eforları için gösterge sayılabilir. Büyük sayıda metot içeren sınıfların üzerinde yapılacak değişiklikler, sınıftan türeyen sınıflar, üzerinde potansiyel etkiye sahiptir. Daha fazla metot içeren sınıflar daha uygulamaya özel ve yeniden kullanımı zor sınıflar olarak görülürler (Dirik, 2011).

- **Kalıtım Ağacının Derinliği (DIT)**

DIT metriği sınıfın türetildiği sınıflarının sayısını gösterir. Sınıf hiyerarşisinin derinliği arttıkça daha fazla metot kalıtım alınır ve sınıfın davranışlarını öngörmek; anlamak zorlaşır. DIT metriğine destekleyici olarak, sınıfın kalıtım aldığı metot sayısı da ölçülebilir.

DIT için önerilen rakam genellikle derinliğin 5' in altında olmasıdır. 5' in üzerindeki derinlikler oldukça karmaşık yapılar doğurabilir. DIT' in 0 olması sınıfın kök olduğunu gösterir. DIT' in ortalama 2-3 arası bir değerde olması yeniden kullanımın iyi seviyede olduğunu gösterir. 2' den küçük derinlikler ise yeniden kullanımın zayıf olduğu alanları işaret eder (Dirik, 2011).

- **Alt Sınıf Sayısı (NOC)**

NOC metriği sınıftan türemiş alt sınıflarının sayısını verir. Alt sınıf sayısı çoğaldıkça kalıtım özelliğine bağlı olarak yeniden kullanımın arttığı anlaşılır. Alt sınıf sayısının çokluğu sınıftan hatalı soyutlama yapıldığını, belki de hatalı bir hiyerarşi kurulduğunu gösterebilir.

NOC, sınıfın nüfus alanı hakkında bir fikir verir. NOC metriği yüksek sınıflar gözden geçirme, test gibi süreçlerin daha dikkatli ve uzun tutulması gereken yerlerdir.

DIT ve NOC metriklerinin kendi içinde getiri ve götürüleri vardır. Yüksek DIT değerleri artan karmaşıklık gösterirken aynı zamanda artan yeniden kullanıma da işaret eder. Aynı şekilde yüksek NOC değeri artan yeniden kullanıma işaret ederken, daha fazla test eforu harcanması gerektiğini gösterir (Dirik, 2011).

- **Nesneler Arası Eşleme (CBO)**

CBO değeri sınıfın kalıtım ilişkisi dışında diğer sınıflar ile kurduğu bağımlılık (eşleme) sayısını gösterir. Örneğin diğer bir sınıfın metotlarının kullanılması ya da diğer sınıfın bir nitelik olarak tanımlanması bunlar arasında gösterilebilir. CBO değeri eşleme yapılmış sınıf sayısını gösterir. CBO değeri sınıfın bağımsızlığının göstergesidir. CBO ve DIT değerleri 0 olan sınıflar tamamen bağımsız olarak çalıştırılabilir sınıflardır. CBO değerinin 5' i aşmaması genellikle tavsiye edilmektedir (Dirik, 2011).

- **Uyumsuzluk (LCOM)**

Bir sınıf elemanının birlikteliğinin derecesini gösterir. Uyumluluk bir sınıfın birden fazla soyutlama yapmamasını gerektirir. Aksi durumda sınıfın başka sınıflara bölünmesi gereklidir. Chidamber ve Kemerer' a göre sınıfın aynı değişkenlerine erişen birden fazla metot varsa sınıfta bir uyumluluktan bahsedilebilir. LCOM bir sınıfın içindeki hiçbir ortak alanı paylaşmayan metot toplamından, en az bir ortak alanı paylaşan metot toplamının farkı ile bulunur. Bulunan değer 0' dan küçükse 0 kabul edilir. LCOM değeri sınıfın metot sayısına bağlı olarak değişmektedir (Dirik, 2011).

3.4. Naive Bayes Algoritması

Büyük miktarda veri içinden, gelecekle ilgili tahmin yapmayı sağlayan bağıntı ve kuralların aranması veri madenciliği olarak adlandırılır (Alpaydın, 2000). Veri madenciliği, karar yapıları, satış stratejileri, finansal tahminler gibi birçok alanda uygulanabilirliğinden dolayı son yıllarda veritabanı uygulayıcılarının ve araştırmacılarının büyük ilgisini çekmektedir. İstatistik, makine öğrenimi, veritabanları gibi tekniklerin kombinasyonu ile veritabanlarındaki kullanışlı ve değerli bilginin ortaya çıkmasına yardım etmektedir (Krishna ve De, 2005).

Veri Madenciliği teknikleri, denetimli ve denetimsiz olmak üzere ikiye ayrılır. Denetimli öğrenmede eğitim verileri vasıtasıyla bir öğrenme işleminin yapılması gerekmektedir. Yani kullanılan eğitim verilerinden bazı kurallar ve etiketler çıkarılarak daha sonraki test sürecinde bu kurallar ve etiketler doğrultusunda sonuçlara gidilir. Veri tabanlarındaki gizli örüntüleri ortaya çıkarmakta kullanılır. Var olan veritabanının bir kısmı eğitim için kullanılır ve böylece sınıflandırma kuralları belirlenir. Bu kurallar yardımıyla yeni bir durum ortaya çıktığında nasıl karar verileceği belirlenir. Denetimsiz öğrenmede ise, denetimli öğrenmedeki eğitim durumu geçerli değildir. Kesinlik içermez. Veriler, kendi aralarındaki benzerlikler göz önüne alınarak gruplandırılırlar. Denetimsiz öğrenme, pazarlama araştırmalarında yaygın olarak kullanılmaktadırlar. Bunun yanı sıra desen tanımlama, resim işleme ve uzaysal

harita verilerinin analizinde kullanılmaktadırlar (Değerli, 2012). Günümüzde veri madenciliği alanında orta çıkmış birçok yöntem bulunmaktadır. Yapay sinir ağları, karar ağaçları, birliktelik kuralları, k-en yakın komşu algoritması ve naive bayes algoritması bunlardan en önemlileridir.

Naive Bayes sınıflandırıcı Bayes teoremine dayanır. Bayes sınıflayıcı, örneklerin hangi sınıfa hangi olasılıkla ait olduklarını tespit eder (Solmaz vd, 2014). Naive Bayes, değişkenler arasındaki ilişkiyi analiz eden tahminci ve tanımlayıcı bir sınıflandırma algoritmasıdır (Değerli, 2012). Naive bayes algoritması, her kriterin sonuca olan etkilerinin olasılık olarak hesaplanması temeline dayanmaktadır. Algoritmanın temeli, verinin öğrenilmesi esasına dayanmaktadır. Eğitimde kullanılan veriler, modelin öğrenilmesi için her çıktının kaç kere meydana geldiğini hesaplar. Bu değer öncelikli olasılık olarak adlandırılır. Hesaplamalar sırasında her bir bağımsız değişkenin bağımlı değişkenlere bölümünün kombinasyonu olayın meydana gelme sıklığını bulur. Bu da veri kümesinden yapılacak tahmin için kullanılır (Akbulut, 2006). Bayes sınıflandırıcı, istatistiksel sınıflandırma teknikleri arasında yer almaktadır. Makine öğreniminde öğreticili öğrenme alt sınıfında yer almaktadır.

Naïve Bayes sınıflandırma yönteminde sisteme belirli bir oranda öğretilmiş veri sunulur. Öğretim için sunulan verilerin mutlaka bir kategorisi bulunmalıdır. Öğretilmiş veriler üzerinde yapılan olasılık işlemleri ile sisteme sunulan yeni test verileri, daha önce elde edilmiş olasılık değerlerine göre işletilir ve verilen test verisinin hangi kategoride olduğu tespit edilmeye çalışılır (Rahman, 2014). Öğretilmiş veri sayısı ne kadar çok ise, test verisinin gerçek kategorisini tespit etmek o kadar kesin olabilmektedir. Eşitlik 3.1' de Bayes sınıflandırmayla ilgili formül verilmiştir (Özkan, 2008).

$$P(X|C_i) = \prod_{k=1}^n P(X_k|C_i) \quad (3.1)$$

Burada X, sınıf üyeliği bilinmeyen veri, C ise sınıf değeridir. Tez çalışmasında naive bayes sınıflayıcısı kullanıldığı için, algoritmayı açıklayan bir örnek aşağıda

anlatılmıştır. Çizelge 3.3’ deki eğitim veri kümesi kullanılarak aşağıda istenilen şartın hangi sınıfa ait olduğu bulunması istenir.

Çizelge 3.3. Örnek veri kümesi

Kilo(kg)	Boy(m)	Yaş	Kabul
65	1.70	28	Bay
54	1.55	26	Bayan
76	1.78	24	Bay
60	1.50	42	Bayan
66	1.48	52	Bayan
84	1.69	55	Bay

Çizelge 3.3’ deki veri kümesi kullanılarak

X_1 : Kilo = 64

X_2 : Boy = 1.71

X_3 : Yaş = 33 verilerine göre örneğin hangi kabule gireceği naive bayes algoritmasıyla aşağıdaki gibi hesaplanmaktadır. İlk aşamada varyans ve ortalama değerler bulunarak Çizelge 3.4’ deki eğitim kümesi oluşturulur.

Çizelge 3.4. Eğitim kümesi

Ortalama	Varyans	Kilo(kg)		Boy(m)		Yaş		Kabul
		75	91	172,3	24,3	35,6	284,3	Bay
		60	36	151	13	40	172	Bayan

Beklenen değeri hesaplamak için test verisinin Bay kısmına veya Bayan kısmına ait olması için beklenen durum hesabı için Eşitlik 3.3’ deki formül uygulanır. Eşitlik 3.4 ise olasılık hesabının sözcükleştirilmiş gösterimidir.

$$Beklenti(Bay) = \frac{P(Bay)P(Boy|Bay)P(Yaş|Bay)P(Kilo|Bay)}{normalleştirme} \quad (3.4)$$

İstenilen şartın bay olma olasılığı için yapılan hesaplama eşitlik 3.5’ de verilmiştir.

$$P(Kilo|Bay) = 0,5 * \frac{1}{\sqrt{2\pi 91^2}} e^{\frac{(64-75)^2}{2*91^2}} = 0,00441725 \quad (3.5)$$

Çizelge 3.7. Hazır Düzenli İfade Desenleri (Devam)

Web Adresi	[a-zA-Z]([a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?\.[a-zA-Z]([a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(\.[a-zA-Z]([a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)?
Tarih :	((([0-9]) ([0-2][0-9]) ([3][0-1]))\/(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec)\/\d{4})
Zaman:	([0-1][0-9] ([2][0-3]):([0-5][0-9]))
E-Posta :	([w-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.) ((\w-+\.+)))([a-zA-Z]{2,4} [0-9]{1,3})(\[[0-9]{1,3}\])?
Guid:	^\{([0,1]([0-9a-fA-F]){8}-([0-9a-fA-F]){4}-([0-9a-fA-F]){4}-([0-9a-fA-F]){4}-([0-9a-fA-F]){12})\}\{0,1\}\$
Ev Telefonu	\d{1}(\d{3})\([-\.s])\d{3}\1\d{2}\1\d{2}

Hazır desenlerde görülen düzenli ifadelerde kullanılan karakter katarları Çizelge 3.7' de verilmiştir.

Çizelge 3.8. Karakter katarları

Karakter	Tanım
\	Kendinden sonraki karakteri özel karaktere dönüştürür veya tam tersini yapar. Örnek: "n" karakter "\n" yeni satır. "(" parantez "\(" karakter
^	Satır başını ifade eder. Örnek: ^Regx , satır "Regx" karakterleri ile başlamak zorunda.
\$	Satır sonunu ifade eder. Örnek: Regx\$ satır "Regx" karakterleriyle bitmek zorunda
*	Kendinden önceki karakterin yada string in hiç olmaması yada istediği sayıda olmasını sağlar. Örnek: "zo*" deseni "z" veya "zoo" ile eşleşir.
+	Kendinden önceki karakterin bir yada daha fazla olmasını zorunlu kılar. Örnek: "zo+" deseni "zoo" ile eşleşir ama "z" ile eşleşmez.
?	Kendinden önceki karakterin olması yada olmamasını sağlar, 0 yada 1 kere. Örnek: "a?ve?" deseni "never" in içindeki "ve" ile uyar. .
.	Herhangi bir karakter yerine geçer.
\b	İfadenin önünde veya arkasında sınırlayıcı (boşluk gibi) karakterleri sınır olarak kabul eder. "\beli\b" önünde ve arkasında boşluk olan "eli" ifadesini ile eşleşir.
\B	\B karakteri sınırlandırma olmayan durumu belirtir.
\d	Sayısal karakter. "[0-9]" desenine eşittir.
\D	Sayısal karakter hariç. "[^0-9]" desenine eşittir.
\n	Yeni satır karakteri

Çizelge 3.9. Karakter katarları (Devam)

\r	Satır başı karakteri
\s	Boşluk , boşluk, tab benzeriyle eşleşir. "[\f\n\r\t\v]" eşittir
\S	Boşluk içermez. "[^\f\n\r\t\v]" desene eşittir
\t	Tab karakter

Çizelge 3.7. Karakter katarları (Devam)

\v	Dik tab karakter
\w	Herhangi bir kelime ile eşleşir. "[A-Za-z0-9_]" desenine eşittir.
\W	Herhangi bir kelime ile eşleşmez. "[^A-Za-z0-9_]" desenine eşittir.
\n	Yeni satır karakteri

Düzenli ifadelerle ilgili .net framework kütüphanesindeki fonksiyonlar aşağıdaki örneklerde verilmiştir.

```
string stringvalue = "guncelsariman@mu.edu.tr 4354564saguncel@yahoo.com Denemeler";
string pattern = "([\\w-\\.]+)@([\\/[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.)([\\w-\\.]+\\.)([a-zA-Z]{2,4})[0-9]{1,3})(\\/?)*";
Regex rgx = new Regex(pattern, RegexOptions.IgnoreCase);
MatchCollection matches = rgx.Matches(stringvalue);
if (matches.Count > 0)
{
    Console.WriteLine("=== Sonuç ===");
    foreach (Match match in matches)
        Console.WriteLine("{0}", match.Value);
}
```

```
string stringvalue = "\"123-456-7890\", \"444-234-22450\", \"940-2036-6578\", \"146-893-232\", \"146-839-2322\", \"4007-295-1111\", \"407-295-1111\", \"407-2-5555\"";
string pattern = "\"(\\d{3}-\\d{3}-\\d{4})\"";
Regex rgx = new Regex(pattern, RegexOptions.IgnoreCase);
MatchCollection matches = rgx.Matches(stringvalue);
if (matches.Count > 0)
{
    Console.WriteLine("=== Sonuç ===");
    foreach (Match match in matches)
        Console.WriteLine("value={0} Groups[1]={1}", match.Value, match.Groups[1]);
}
```

3.6. Web Ortamında Veri Gönderme Yöntemi

Http, sunucu ve istemcinin birbiri ile iletişimi için tasarlanmış bir iletişim protokolüdür. Http, istek ve cevap kuralı ile çalışır. Bu protokolda “get, post, put, trace, delete, head ve options” olmak üzere 7 adet istek tipi bulunmaktadır. Get

ve Post istek tipleri, verilerin sunucu ve istemci arasında haberleşmede en sık kullanılan yöntemlerdir. Get metodunun özellikleri aşağıdaki gibi listelenmiştir.

- ✓ *Kaynaktan veri talebinde bulunur.*
- ✓ *İstek önbelleklenebilir (caching).*
- ✓ *İstek tarayıcı geçmişinde kalır.*
- ✓ *Yerimi olarak kaydedilebilir.*
- ✓ *Hassas verilerle çalışırken kullanılmamalıdır.*
- ✓ *İsteklerin veri uzunluk sınırlamaları vardır.*
- ✓ *Sadece veri alınırken kullanılmalıdır.*

GET metodu genellikle tarayıcıların adres çubuğundaki isteğin 512 karakteri geçmediği yerlerde HTML sayfası almak veya bir resim görüntülemek istendiğinde kullanılır. GET metodu ile yapılan istekler tarayıcının adres satırında görünür. Sadece belirli boyutta veri gönderileceği zaman kullanılır. Form parametreleri Sorgu Dizesi (QueryString) taşınırken adres çubuğunda görülmesi dikkat edilmemesi durumunda ciddi güvenlik zafiyetlerine neden olabilmektedir (Karadeniz, 2012).

Asp.Net de *Request.QueryString["yazar"]*
Php dilinde ise *\$yazar=\$_GET['yazar']*

Post metodu ise aşağıdaki özellikleri barındırmaktadır.

- ✓ *İstek önbelleklenemez.*
- ✓ *İstek tarayıcı geçmişinde kalmaz.*
- ✓ *Yerimi olarak ayarlanamaz.*
- ✓ *Veri uzunluğu kısıtı yoktur.*

POST metodu; URL' de görünmesi istenmeyen veriler olduğunda, dosya yükleme durumlarında, querystring' in çok büyük olduğu durumlarda kullanılır. İstek yaparken gönderilen parametreler tarayıcıların desteklediği formatlarda olmayabilir. Japonca bir kelimeyi ya da Arapça bir harfi parametre olarak gönderirken, tarayıcılar bu karakterleri anladıkları formata çevirirler. Bu da genellikle uzun bir ifadeye karşılık gelir. Normalde kendi dilinde kısa olan bir kelime, parametre geçilirken URL' nin büyümesine sebep olur. Bu gibi durumlarda da POST metodunun kullanılması önerilmektedir. POST yönteminde gönderebilen verinin boyutu ile ilgili bir sınır yoktur. Gönderilen parametrelerin

adres satırında görünmemesi nedeniyle GET yöntemine göre daha güvenlidir. HTML form gönderileceği zaman neredeyse her zaman POST yöntemi tercih edilir (Pehlivan, 2014).

3.6.1. Asp.Net teknolojisinde veri aktarma yöntemleri

Asp.net kullanıcı istekleri arasında verilerin iletimi için farklı yollar sunmaktadır. Uygulama nesnesi, cookie, hiddenfield, Session, Cashe nesnesi ve diğer birçok yöntem kullanılabilir. ASP.NET’ te verileri saklama veya sayfalar arası taşıma işlemleri, Durum Yönetimi adı altında sunulan seçenekler kullanılarak yapılır.

Uygulamaların tamamı içinde yer alan nesneler, değişkenler, özel olarak tanımlanan her şey Application adı verilen nesne içinde saklanabilir. Bu nesne içinde saklanan her türlü veriye Application State adı verilir. Application State ayrıca uygulamalarda yer alan tüm sayfalar içinde bu nesne içindeki verileri paylaşma imkânı tanır.

```
Application["SiteName"]="Güncel Sarıman";  
string appName = (string)Application["SiteName"];  
Application.Remove("SiteName");
```

Application state, session state ile oldukça benzerdir. Session state kullanıcının ziyareti süresince ilgili değişkenleri ve verileri saklar. Application state ise değişkenleri ve nesneleri aynı anda uygulamanın tüm kullanıcıları tarafından paylaşımına sunmaktadır. Böylelikle tüm kullanıcıların ortak olarak kullanmış olduğu bir uygulamada verileri saklamak için en ideal durum application state olmaktadır (Gürsoy, 2011).

Bazı kaynaklarda, HTTP Cookie, Web Cookie, Browser Cookie, Session Cookie olarak da bilinen çerezler, internet sitelerinin bilgisayarlara kaydettiği küçük metin dosyalarıdır. Genel kullanım amacı, siteye gelen ziyaretçileri, bir sonraki ziyaretlerinde hatırlamaktır. Çerezler, web sunucusu tarafından tarayıcıya gönderilen ve istemcide kaydedilen ufak metin dosyalarıdır.

```
Response.Cookies["Musteri"].Value="güncel"  
string adsoyad = Request.Cookies["Musteri"].Value;
```

Çerezler, kullanıcı kimlik doğrulama ve tanımlama, kullanıcı tercihleri, alışveriş sepeti, vb. bilgileri istemci tarafında tutmak için kullanılır. Ayrıca, bilgilerin bir sayfadan başka bir sayfaya taşınması için de kullanılır. Çerezde tutulan bilgiler, kullanıcının hesabına giriş yapan biri tarafından rahatlıkla çalınabilir (Altinkurt, 2012).

Hidden Field form nesnesinin kullanılmasının en büyük sebebi verilerin Framework ile geliştirilen uygulamalar arasında transfer olmasıdır (Smith, 2014). ASP.Net Hidden Form Fields, açılan sayfada gözükmeyen alanlardır. Sayfanın kaynak kodunun içerisinde, değeri ile birlikte görünür. Gizli alanlar sayfalar arasındaki geçişlerde çeşitli bilgilerin saklanması için kullanılabilmektedir.

Sayfalar arası veri taşıyabilmek için QueryString yöntemi de kullanılmaktadır, URL üzerindeki stringler ile verileri taşımaya sağlayan QueryString fazla değerli verilerin olmadığı anlarda tercih edilmesi gereken yöntemdir. Olumlu yanı, URL adresinin içinde bulunmasından dolayı, sayfanın yer imlerine kaydedilmesi durumunda kendisinin de saklanmasıdır. Böylece yer imine gidildiğinde, sayfa saklandığı haldeki şekliyle açılır.

```
Response.Redirect("Bilgiler.aspx?KullaniciAdi="+txt_kullaniciAdi.Text+"&Sifre="+  
txt_Sifre.Text+"");
```

Yukarıdaki örnek kod satırındaki Bilgiler.aspx kısmı bilgilerin hangi sayfaya gönderileceğini söylemektedir (Yazıcıoğlu, 2007).

Session, herhangi bir web sunucusuna bağlanıldığında kullanıcılara verilen bir çeşit kimliktir. Session kelime anlamı olarak oturum demektir ve bir oturum süresince kullanılabilmektedir. Oturum süresi bir tarayıcının başlatılmasıyla kapatılması arasında geçen süre kadardır. Sayfalar arasında veri taşıyabilmektedir. Kullanıcılara ait kullanıcı adı, şifre gibi bilgilerin sunucu

üzerinde kaydedilmesini sağlayan bir yapıdır. Session zaman aşımına uğrayınca veya kullanıcı siteden çıkış yaptığında o kullanıcı için silinir, sitedeki diğer kullanıcıların bilgileri saklı tutulmaya devam eder. İçerisindeki veri sunucu tarafında tutulduğu için en güvenli yöntemlerden biridir. Herhangi bir veri kullanıcı bazlı ve gizli tutulması gerekiyor ise bu yöntem tercih edilmelidir. Asp.net ile session tanımlama iki şekilde yapılabilir.

```
Session.Add("SessionAdiniz","GirmekIstediginizDeger");  
Session["SessionAdiniz"] = "GirmekIstediginizDeger";(Güvercin, 2007)
```

Performans için çok önemli bir konu olan ön belleğe alma mekanizmasının Output Caching yöntemi ile sayfanın çıktısı ön belleğe alınabilmektedir. Output Caching ile sayfa çıktısının tamamı ya da belli bir kısmı ön belleklenebilir. Bazı durumlar da sayfa içerisinde kullanılan verinin de ön belleklenmesi gerekebilir. Bu durumda Output Caching kullanılamaz. Uygulama içerisinde veri ya da verileri ön belleğe alabilmek için veri ön belleğe alma yöntemi kullanılabilir. İstenilen bilginin tekrar tekrar kaynaklardan çekilmesi yerine, gelen ilk istek sunucu tarafından karşılanacak sonrasında ise ön belleklenen veri bellekten alınıp uygulama içerisinde kullanılacak. Veritabanından alınan verilerde bu işlem performans için çok önemli bir artış sağlamaktadır (Göksu, 2009).

```
Cache.Insert("Key", Data);  
Cache.Remove("Key");  
Cache.Insert("Listem", Liste, null, DateTime.Now.AddYears(2),  
System.Web.Caching.Cache.NoSlidingExpiration);
```

Context nesnesi veriyi tek kullanıcı adına tutar ve istek süresince saklanır. Context taşıyıcısı büyük verileri saklar fakat genel olarak ufak verileri tutmak için kullanılmaktadır.

View State Türkçe olarak durum yönetimi olarak bilinir. Klasik internet çalışma prensiplerinde sayfa yeniden yüklediğinde formlardaki veriler sıfırlanır. Ancak ASP.NET ViewState yapısı sayesinde, yeniden yüklenme olması durumunda bile web sayfalarındaki kontrollerin taşıdıkları verileri saklar. ViewState içeriğinin kullanıcılar tarafından görülme olasılığı yoktur. Viewstate içeriği sayfada gizli olarak saklanır. ViewState sayfa üzerindeki bilgileri sakladığı için sayfanın

boyutunda bir artış meydana gelmekte ve bu durumda performans kaybına sebep olabilmektedir. Bazı sayfalarda ise durum bilgisinin saklanması gerekebilir. İstenilen herhangi bir sayfada ViewState özelliği kapatılabilir. `EnableViewState = "False"` özelliği False yapılarak, o kontrol üzerinde ViewState kullanılması önlenmiş olur. ViewState değerlerine ulaşmak için ViewState üzerinde değişkenler oluşturularak ViewState' ler hakkında bilgi sahibi olunabilmektedir

ViewState ["ornekisim"]=txt_kullaniciadi.Text

3.6.2. Php' de veri aktarma yöntemleri

PHP çalıştırdığı her betiğe çok sayıda önceden tanımlanmış değişken sağlar. Değişkenler dış kaynaklı değişkenlerden yerleşik ortam değişkenlerine, son hata iletisinden son elde edilen başlıklara kadar geniş bir sahada kendilerini gösterirler (Php Group, 2015).

Global tanım kapsamında tanımlanmış bütün değişkenleri barındıran \$GLOBALS dizisi sayfada herhangi bir noktadan bu değişkenlere ulaşmak için kullanılabilir. PHP 3.0.0 sürümünden beri kullanılabilen \$GLOBALS dizisine değişken tanımlamaları global anahtar sözcüğü ile yapılabilir. Ancak güvenlik açıkları sebebiyle bu yolun kullanımı önerilmemektedir (Daylak, 2014).

```
<?php
function Tut(){
    $GLOBALS['A']=100;
    $B=500;
}
Tut();
echo $GLOBALS['A'];
echo "$B";
?>
```

\$_GET değişkeni, get metoduyla veri alan formlardaki verileri ele alır, HTTP GET metodu değişkenler dizisinin bir elemanıdır. Bir formdan gönderilen bilginin kullanıcı tarafından erişilebilmesi için GET metodu kullanılır. \$_GET değişkeni kullanıldığında bütün değişken isimleri ve değerleri URL' de görüntülenir. Şifre işlemleri ve özel veri aktarımı için bu method güvenli değildir (Akoğlu, 2008).

Merhaba <?php echo \$_GET["name"]; ?>.

\$_POST değişkeni, method="post" yöntemiyle bir formdan gönderilen verileri toplar. POST metoduyla gönderilen bilgilerin büyüklük sınırı yoktur ve kullanıcı tarafından görülemezler.

```
<?php
echo 'Merhaba '. htmlspecialchars($_POST["isim"]). '!';
?>
```

PHP’de dosya yükleme işlemleri \$_FILES dizi değişkeni ile yapılır. Gönderilen dizi değişkeninin adı dosya olduğu için \$_FILES['dosya']['dizi_ozelligi'] şeklinde kullanılır.

```
<form action="yukle.php" method="post" enctype="multipart/form-data">
  <input type="file" name="upload" />
  <input type="submit" value="Gönder" />
</form>
<?php $upload = diverse_array($_FILES["upload"]); ?>
```

\$_GET, link satırından gelen istekleri almaya yarayan bir methodken, \$_POST, form alanından gönderilen verileri almaya yarar. \$_REQUEST ise \$_GET ve \$_POST diye ayırmadan, form olmaksızın sayfaya gönderilen bilgileri almaya yaramaktadır. \$_GET, \$_POST ve \$_COOKIE değişkenlerini içeren ilişkisel bir dizidir

```
<form action="hosgeldin.php"> Adınız:<input type="text" name="ad" /> <input type="submit"
value="Gönder" /> </form>
<?php
echo "HOŞGELDİNİZ ".$_REQUEST["ad"];
?>
```

Php dilinde sessionlar tek bir \$_SESSION global dizisi üzerinden okunup silinebilmektedir. Session kullanımına ya da oluşturulmasına başlamadan önce bir kere her sayfanın en başında oturumu başlatmak adına session_start() fonksiyonu yazılmalıdır.

```
$_SESSION['mesaj'] = 'Selam dünya!';
```

PHP’de cookie oluşturma işini bir setcookie() fonksiyonu ile yapılır. Fonksiyonda tarayıcıya gönderilecek ve saklanacak olan değerın anahtarını ve içeriği belirtildiği gibi bir de ne kadar süre tarayıcıda saklanacağını da belirtilir.

```
setcookie("çerezim", "selam dünya");
echo $_COOKIE['çerezim']
```

3.6.3. Java’ da veri aktarım yöntemleri

Java web programlamada kullanılan jsp ve jsf teknolojileri session, querystring, cookie gibi veri transferi ve saklama yöntemlerinde Asp.Net ve Php dillerinde olduğu gibi benzer kavramlar kullanılmaktadır.

Javanın farklı sürüm ve teknolojilerinde sayfalar arası aktarılan verileri almak ve kaydetmek için “getParameter, getParameterValues, getHeader, getHeaders, getContextPath, getQueryString, getSession” fonksiyonları kullanılmaktadır.

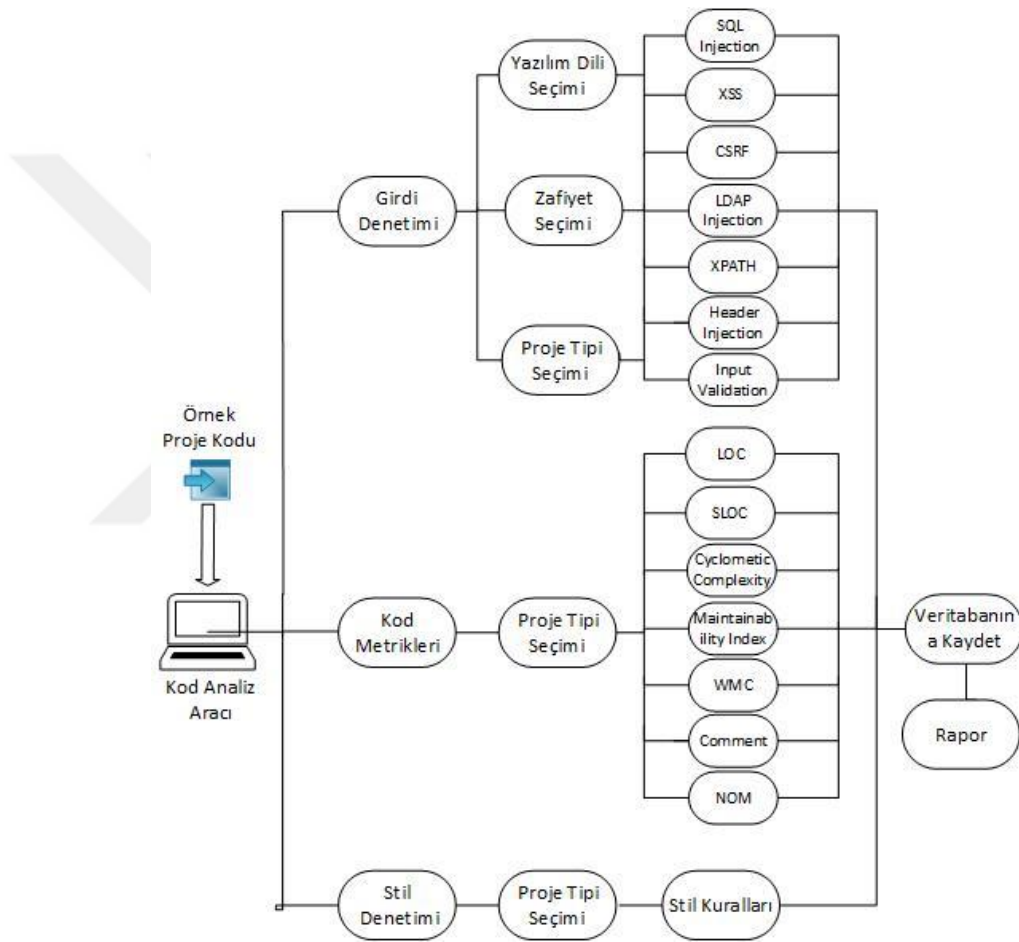
3.7. Yöntem

Gerçekleştirilen tez çalışmasında web projelerindeki güvenlik zafiyetlerinin ortaya çıkartılması amaçlanmıştır. Artan web teknolojilerinin kontrolsüz büyümesi beraberinde zafiyet barındıran önemli uygulamaların kullanıcılara sunulmasına ve denetimsizliğe neden olmaktadır. Bu çalışma, geliştiricilerin henüz yazılımı devreye almadan önce yazılımdaki güvenlik açıklarını önlemeyi sağlayacak bir uygulamayı kapsamaktadır. Çalışmanın ilk bölümünde geliştiricilerin kapsamlı bir uygulama denetimi yapabileceği kod analiz aracı geliştirilmiştir. İkinci bölümde ise kod metrikleri kullanılarak web uygulamalarındaki güvenlik seviyesini ölçen bir web uygulaması gerçekleştirilmiştir.

3.7.1. Kaynak kod analiz yazılımı

Kaynak kod analiz yazılımı, web projelerindeki muhtemel güvenlik açıklarının henüz proje derlenmeden ve hizmete alınmadan önce proje geliştiricileri tarafından kullanılabileceği bir yazılımdır. Geliştirilen yazılım java, asp.net ve php dillerinde geliştirilen uygulamaların derlenmeden, saf kod halinde analizini gerçekleştirebilmektedir. Statik analiz yöntemlerini kullanarak geliştirilen yazılım masaüstü uygulaması olarak geliştirilmiştir. Günümüzde yaygınlaşan web uygulamaları hesaba katıldığında projenin web tabanlı olması gerektiği düşünülebilir. Fakat proje geliştiricileri için önemli olan gizlilik ve mahremiyet,

geliştiricilerin kodlarını internet sitesine yükleyerek analiz yapmalarını imkânsız kılmaktadır. Geliştiriciler için hayati önemi olan kaynak kodların kimseyle paylaşılmaması ve hatta çalıştıkları kurumda çevrimdışı bilgisayarlarda kod geliştiriyor olmaları gizliliği en üst seviyeye çıkarmaktadır. Bu sebeple geliştirilen proje masaüstü uygulaması ve offline çalışabilme özelliğine sahiptir. Kod analiz aracı 3 bölümden oluşmaktadır. Şekil 3.3’ de kaynak kod analiz yazılımının blok diyagramı verilmiştir.



Şekil 3.3. Kaynak kod analiz yazılımı blok diyagramı

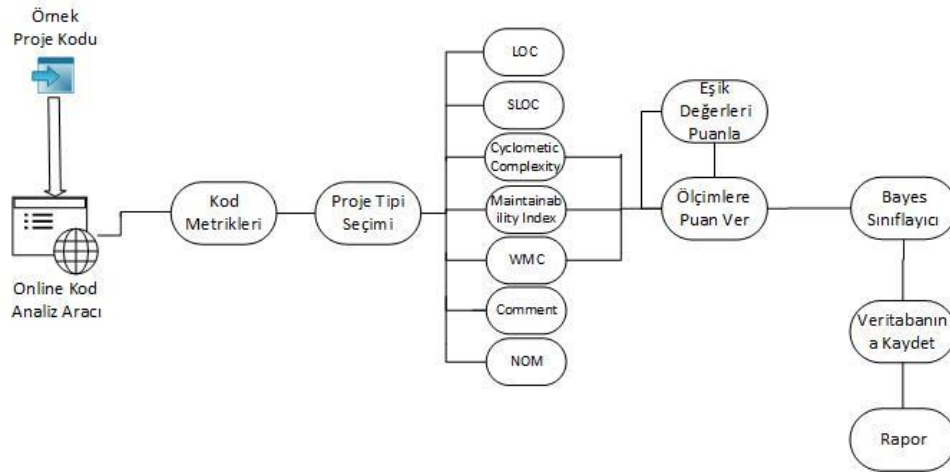
İlk bölümde zafiyetleri oluşturabilecek girdilerin projelerdeki denetimleri yapılmaktadır. İkinci bölümde ise kod kalite ölçümleri için kaynak kod metriklerinin ölçümü gerçekleştirilmektedir. Son bölümde ise projelerdeki stil denetimleri gerçekleştirilebilmektedir. Analiz aracı .Net Framework 4.5

kullanılarak Visual Studio derleyicisinde geliştirilirken, veritabanı olarak Ms Sql Server kullanılmıştır.

Statik kod analizi yapılarak geliştirilen yazılımda kaynak kodların saf hali kullanılarak analiz gerçekleştirilmektedir. Girdi denetiminde projelerdeki kodların satır satır ele alınarak incelenmesiyle analiz gerçekleştirilebilmektedir. Zafiyet türlerine göre incelemesi yapılan satırlarda aranan desen bulunarak üst veya alt satırlardaki durumu kontrol edilmektedir. Desen araması yapılırken düzenli ifadelerden yararlanılmaktadır. Böylece daha hızlı ve kesin arama sonuçları elde edilebilmektedir. Kod metriklerinin ölçümü sırasında analiz aracına yüklenen projenin tüm kodları ayrıştırılarak metrik türüne göre hesaplamalar yapılmaktadır. Projedeki her kod dosyası kendi içerisinde değerlendirilmektedir. Stil denetimi için 3 programlama dilindeki kodlama standartları incelenerek, kategorilerine göre uyulması istenen kurallar yüklenen proje dosyalarında aranmaktadır.

3.7.2. Yazılım güvenlik seviyesi belirleme aracı

Yazılım güvenlik seviyesi bulma aracı, geliştiricilerin henüz proje derlenmeden önce kullanabileceği bir yazılımdır. Hesaplanan metrikler yazılım kalite ve sürdürülebilirliği açısından geliştiricilere fikir verebilmektedir. Güvenli bir yazılım geliştirmek için yazılımın kalite seviyesi yüksek olmalıdır. Geliştirilen metodlardaki ağırlık, operator, satır sayısı gibi faktörler projenin yavaşlamasına, hatalı çalışmasına hatta durmasına yol açabilmektedir. Yapılan literatür araştırmalarında çevrimsel karmaşıklık düzeyinin güvenliği doğrudan etkilediği gözlemlenmiştir. Karmaşık sistemler çok fazla kod satırı içerebilirler, böylece güvenlik açıkları oluşabilir. Karmaşık sistemleri test etmesi oldukça zordur ve test edilemeyen yazılımlar zafiyet barındırabilirler. Karmaşıklık hakkındaki tüm bilgiler güvenliği etkilemektedir (McCabe, 2008). Şekil 3.4' de yazılımın blok diyagramı verilmiştir.



Şekil 3.4. Yazılım güvenlik seviyesi belirleme yazılımı

Kaynak kod metriklerinin yazılım projeleri üzerindeki önemi ve kod kalitesinin güvenlik üzerinde dolaylı yoldan olan etkisiyle metrikler kullanılarak web tabanlı bir güvenlik ölçüm aracı geliştirilmiştir. Analiz aracının web tabanlı geliştirilmesi çalışmanın eğitimsel açıdan tüm kullanıcılara yararlı olmasını amaçlamaktadır. Yazılım, bilgi güvenliği ve yazılım güvenliği alanında çalışan veya eğitim alan kişiler için bu yazılım aracının yol gösterici olması düşünülmüştür. Kod metriklerinin hesaplanması statik kod analiz çalışmalarının bir parçasıdır.

Nesneye yönelik metrik türünün değerlendirmeye alındığı yöntemde, ölçülen metriklere, literatürde belirlenen eşik seviyelerine göre belirlenen puanlar verilmiştir. Güvenlik derecelendirilmesi ile projedeki her kod dosyasına ölçülen metrik değerinin puanlamasına göre puan verilmiştir. Makine öğrenme algoritması kullanılarak projedeki tüm kod satırı barındıran dosyaların güvenlik seviyesine göre sınıflandırması yapılmıştır. Çalışmada sınıflandırma algoritması olarak Naive Bayes sınıflandırma algoritması kullanılmıştır. Sınıflandırması yapılan projeler, geliştirici tarafından süreç boyunca takip edilebilmektedir. Güvenlik analiz aracı Asp.Net teknolojisiyle ve Ms Sql Server veritabanı kullanılarak geliştirilmiştir.

4. ARAŞTIRMA BULGULARI

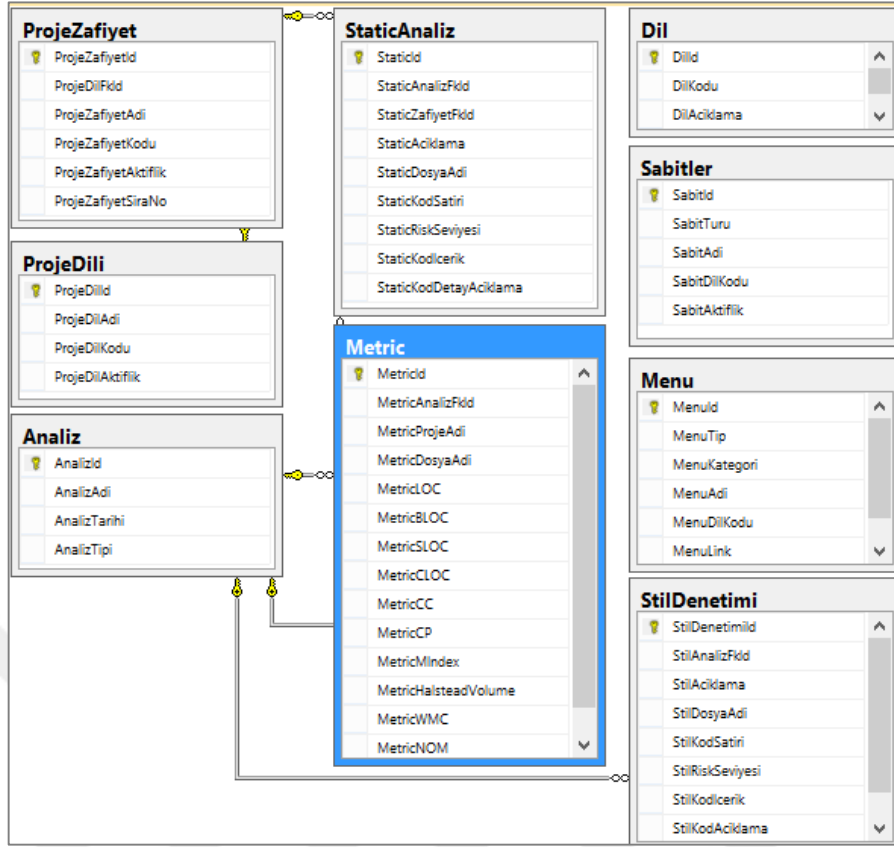
Tez çalışmasında yazılım test ve değerlendirme aracı geliştirilmiştir. Çalışma iki bölümden oluşmaktadır. İlk bölümde kaynak kod analiz yazılımı, ikinci bölümde ise güvenlik seviyesi belirleme yazılımı geliştirilmiştir. Her iki yazılım da statik kod analiz teknikleri kullanılarak gerçekleştirilmiştir. Çalışma kapsamında piyasada kullanılan açık kaynak yazılımlar test edilmiş ve sonuçları araştırma bulguları bölümünde verilmiştir.

4.1. Kaynak Kod Analiz Yazılımı

Kaynak kod analiz yazılımı Wpf teknolojisi kullanılarak Visual Studio 2013 ortamında masaüstü uygulaması olarak geliştirilmiştir. Analiz aracı, yazılımların güvenli bir şekilde geliştirilebilmesi için programcılara yol gösterici olması amacıyla gerçekleştirilmiştir. Analiz aracı programcıların, yazılım süreci boyunca gerçekleştirilen analiz aşamalarını kaydederek projedeki güvenlik zafiyetlerinin giderilme oranını izlemelerini sağlamaktadır. Yazılımda geriye dönük raporlama alınarak geliştiricinin kodlama seviyesi hakkında bilgiler de çıkarılabilmektedir. Geliştirilen yazılımı test etmek için piyasada sık kullanılan açık kaynak kodlu yazılımlar ve örnek projeler analiz edilmiştir. Kaynak kod analiz yazılımı, test projelerini 3 aşamada değerlendirmektedir. Web uygulamalarındaki güvenlik zafiyetlerini engellemek adına girdi analizi, kaynak kod kalitesini bulmak için metrik analizi ve stil denetimleri için stil denetleyici bölümleri yazılımı oluşturan bölümlerdir. Çalışmanın ilerleyen aşamalarında bu bölümler sırasıyla anlatılacaktır.

4.1.1. Yazılım veritabanı

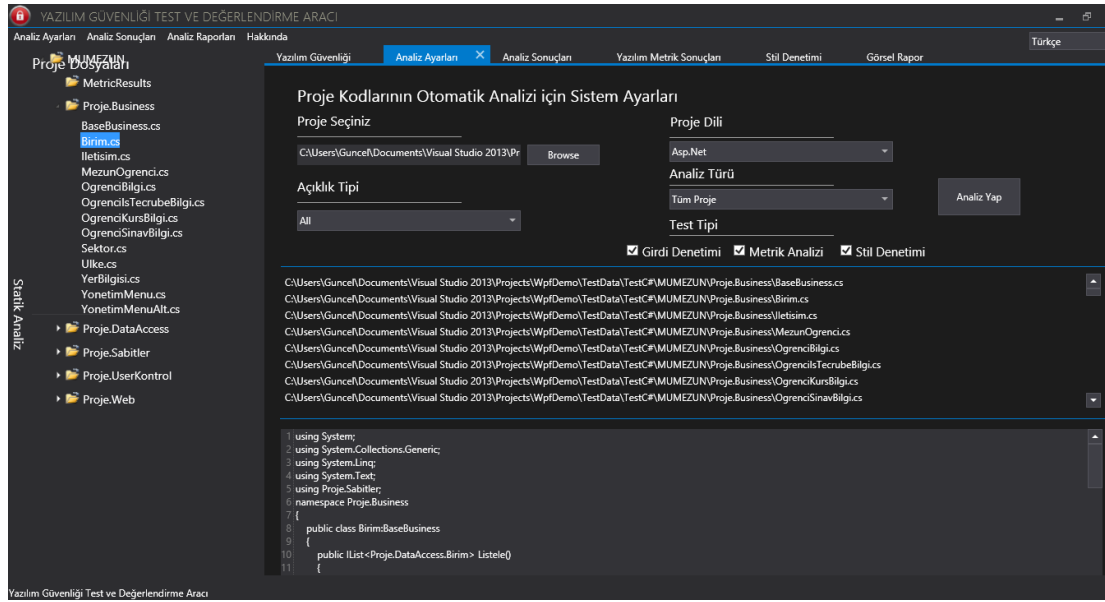
Geliştirilen yazılım kullanıcılara Türkçe ve İngilizce dil seçenekleri sunmaktadır. Ayrıca java, asp.net ve php dillerinde geliştirilen projeler için her dildeki zafiyet türlerinin tek tek analizi ve projedeki dosyaların bireysel analizi için yazılım arayüzünde farklı seçenekler sunulmuştur. Şekil 4.1' de kaynak kod analiz yazılımına ait veritabanı diyagramı verilmiştir.



Şekil 4.1. Kaynak kod analiz yazılımı veritabanı diyagramı

4.1.2. Girdi denetimi

Yazılım güvenliği test ve değerlendirme aracının girdi denetimi sekmesinde kullanıcılar analizini gerçekleştireceği projeyi gözet düğmesine tıklayarak yükleyebilmektedirler. Proje dili seçeneği yardımıyla Java, asp.net ve php dillerinden biri seçilebilir. Arayüzdeki açıklık tipi seçeneği, seçilen dile göre değişiklik gösterebilmektedir. Proje kapsamında java, php ve asp.net' de en sık görülen ve önemli olan zafiyetler açıklık tipi olarak seçilebilmektedir. Arayüzün sol bölümünde yüklenen proje ağaç yapısında gösterilmektedir. Herhangi bir dosya seçilerek dosya içeriği aynı arayüzde incelenebilmektedir. Analiz türü seçeneği ile tekli dosya veya tüm projenin girdi denetimi yapılabilir. Şekil 4.2' de yazılım güvenliği test ve değerlendirme aracına ait girdi denetimi arayüzü gösterilmektedir.



Şekil 4.2. Yazılım güvenliği test ve değerlendirme aracı girdi denetimi arayüzü

Girdi denetimi analizinde web uygulamalarının farklı dillerde girdi kabul etme yöntemleri dikkate alınmıştır. Java, asp.net ve php dillerindeki veri alışveriş ve işleme yöntemi güvenlik zafiyetlerinin en önemli sebepleridir. Çalışma kapsamında geliştirilen girdi denetiminin çalışma mantığında öncelikle kullanıcıdan alınan proje dosyalarının işlenmesi aşaması bulunmaktadır. Projelerin içerisindeki resim, video gibi analize gerek duyulmayan dosya tipleri çıkarılarak filtreleme gerçekleştirilir. Arındırılan dosyaların analizi için her bir programlama dilinde girdi içeren satırlar tespit edilmeye çalışılır. Web uygulamalarındaki girdi kabul eden alanlar tespit edildikten sonra zafiyetlerin çeşitlerine göre veri gönderme yöntemleri ve kullanıldıkları satırlar üzerinde incelemeler yapılmaktadır.

4.1.2.1. Asp.net teknolojisinde incelenen zafiyetler

Asp.Net ile geliştirilen uygulamalarda sık rastlanan zafiyetler ve zafiyet oluşturabilecek kodlama problemleri şu şekildedir;

- *Sql Enjeksiyonu*
- *Xss (Çapraz Betik Sorgulaması)*
- *Girdi Doğrulama*
- *TocTou (Time of Check)*
- *Güvensiz Kod*
- *Xpath Enjeksiyonu*
- *Ldap Enjeksiyonu*

- Header Enjeksiyonu
- Asp.Net Config Dosyası Saldırısı
- Temp Dosyalarının Zafiyeti
- Yönlendirme Hataları
- Kayıt (Log) Kontrolü
- Serileştirme Problemleri

Asp.Net ile web programlama yapılırken sayfalar arası veri transferini yapan

- *Request.QueryString*,
- *Request.Cookies*
- *Request.Form*
- *Request.ServerVariables* nesneleri geliştirilen yazılımda kontrol edilmektedir.

Asp.Net teknolojisinde UserControl nesneleri ile de sayfa tasarımı yapılabilmektedir. Sayfaların uzantısı .ascx dir. Sayfa tasarımında .aspx ve .ascx uzantılı sayfalardaki inputlar tüm projede taranarak tespit edilir. Böylece zafiyet barındıran kod satırlarının tespitine başlanmış olunur.

SQL Enjeksiyonu, SQL sorgusu oluştururken kullanılan parametre değerlerinin, güvenilmeyen kaynaklar tarafından kontrol yapılmadan kullanılmasıdır. Veritabanı bağlantısı olan kod satırlarında girdi denetiminin yapılması gerekmektedir. 3 farklı dilde sql enjeksiyonunu engellemek için geliştirilen sistemin çalışma altyapısı Çizelge 4.1' de verilmiştir.

Çizelge 4.1. Sql enjeksiyonu tespit etme çalışma altyapısı

No	Sql Enjeksiyonu Uygulama Adımları
1	Proje Sayfalarındaki Girdi Kontrolleri taranır.
2	Okunan Satırlarda veritabanı sorgusu taranır.
3	Veritabanı sorgusu bulunan satırlar parametre içeriyorsa bu parametreler satırlardan ayıklanarak kaydedilir.
4	Bulunan parametrelerin Girdi nesnesi veya Request parametresi olup olmadığı kontrol edilir.
5	Eğer Girdi nesnesi veya request parametresi ise temizleme fonksiyonundan geçip geçmediği alt ve üst satırlarda kontrol edilir.
6	Temizleme fonksiyonundan geçen parametre güvenli olarak işaretlenir.
7	Girdi nesnesi veya request parametresi değilse üst satırlarda o parametre üzerinde temizleme fonksiyonu kullanılmış mı?
8	Temizleme fonksiyonundan geçen parametre güvenli olarak işaretlenir. Tüm satırlarda hiçbir işlem yapılmadıysa Sql Enjeksiyonu zafiyeti vardır.

Çizelge 4.1. Sql enjeksiyonu tespit etme çalışma altyapısı (Devam)

9	Satırda Sorgu yok ise satırda Girdi alan yerler tespit edilir.
10	Girdilerin kullanıldığı yerde veya alt satırlarda temizleme işleminden geçip geçmediği kontrol edilir.
11	Kontrol işleminin sonucuna göre sql enjeksiyonu vardır.

Asp.Net uygulama dilinde “select * from”, “update”, “set”, “delete from”, “insert into” gibi veritabanı ifadeleri geliştirilen projede kontrol edilmektedir. Bu ifadeler hem klasik sql hemde LINQ sorgulama dilinde kullanılmaktadır. Zafiyetlerin meydana gelebileceği satırlar tespit edilirken zafiyetler risk seviyesine göre kategorize edilmektedir.

XSS Saldırısı, kullanıcı kontrolündeki girdinin, üzerinde denetim uygulanmadan tekrar kullanıcıya gönderilmesidir. Uygulama sunucuda çalışabileceği gibi, tarayıcıda çalışan bir javascript uygulaması da olabilmektedir. Çalışma kapsamında geliştirilen kaynak kod analiz yazılımı XSS saldırısını incelerken veritabanı sorgularının olduğu bölümleri atlayarak analiz yapmaktadır. Sql sorgularının bulunduğu satırlar muhtemel SQLI saldırı satırı olarak kabul edilmiştir. Çizelge 4.2’ de XSS saldırısının uygulama adımları verilmiştir.

Çizelge 4.2. Xss saldırısı tespit etme sistemi çalışma altyapısı

No	XSS Saldırısı Uygulama Adımları
1	Projedeki aspx ve ascx uzantılı sayfalarındaki Girdi ve Çıktı Kontrolleri taranır.
2	Okunan Satırlarda veritabanı sorgusu bulunan satırlar dışındaki satırlar taranır.
3	Satırda Temizleme işlemi varsa o satır atlanır.
4	Satırda Temizleme işlemi yoksa Response.Write veya Response.Form kontrollerinin olup olmadığı kontrol edilir.
5	Varsa bu kontrollerin aldığı parametreler kontrol edilir.
6	Aldığı parametreler Girdi veya Request Nesnesi ise ve Temizleme işleminden geçirilmediyse XSS zafiyeti vardır.
7	Parametre normal bir parametreyse üst satırlarda temizleme işleminden geçip geçmediği kontrol edilir.
8	Satırda .Text, .Content, InnerHtml varsa ve eşitliğin sol bölümünde ise aldığı değer kontrol edilir.
9	Sağ bölümde aldığı değer Request nesnesi veya Girdi mi, Parametre mi olup olmadığı kontrol edilir.

Çizelge 4.2. Xss saldırısı tespit etme sistemi çalışma altyapısı (Devam)

10	Request veya Girdi ise temizleme işleminden o satırda geçmiş mi o kontrol edilir. Temizleme işleminden geçmediyse XSS zafiyeti vardır.
11	Parametre ise üst satırlarda kontrolden geçmiş mi kontrol edilir. Temizleme işleminden geçmediyse XSS zafiyeti vardır.
12	Satırda document.write fonksiyonu var ise ve parametre alıyorsa
13	bu kontrolün aldığı parametreler kontrol edilir.
14	Aldığı parametreler Girdi veya Request Nesnesi ise ve Temizleme işleminden geçirilmediyse XSS zafiyeti vardır.
15	Parametre normal bir parametreyse üst satırlarda temizleme işleminden geçip geçmediği kontrol edilir.

Asp.Net ile geliştirilen uygulamalarda Dom Based Xss saldırısının yapılabileceğinin tespiti satırda “document.write” ifadesinin geçmesiyle bulunabilmektedir. Parametrelerin bulunduğu yere göre risk kategorisi belirlenmektedir.

İstek Doğrulama, asp.net teknolojisinde html verilerin sunucuya kaydedilmesi sırasında muhtemel saldırılara karşı sunucu ve istemciyi korumayı sağlayan bir kontoldür. Çizelge 4.3’ de tez çalışması kapsamında kontrolü sağlanan isteklerin kaynak kodlar üzerinde koruma aşamaları gösterilmiştir.

Çizelge 4.3. İstek doğrulama tespit etme sistemi çalışma altyapısı

No	Validate Request Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Eğer dosya uzantısı .config ise
3	<pages validaterequest değeri aranır.
4	Değer=true ise validation kontrolü vardır.
5	Validaterequest değeri yoksa sayfa ve proje muhtemel saldırılara açık olabilmektedir.
6	Eğer dosya uzantısı .aspx .ascx .asp ise @page control validaterequest satırı kontrol edilir.
7	Kontrol değeri true olarak ayarlanmış ise validation kontrolü vardır.
8	Validaterequest değeri yoksa sayfa saldırılara açık olabilmektedir.

Kaynak kod analiz yazılımında, asp.net ile geliştirilen uygulamalar arasında kontrol edilen bu doğrulama yöntemi, XSS ve bunun gibi birçok saldırıyı engelleyebilmektedir.

Kontrollerin Kullanım Zamanı Saldırısı (TocTou-Time of Use & Time of Check), kodlama esnasında kullanılan kontrollerin eşitlenme ve kullanılma zamanlarının tutarsızlığından dolayı oluşabilen yazılım problemidir. TocTou oturumlar arası yapılan işlemlerde aldatmaya neden olmaktadır. Genellikle dosya okuma ve yazma işlemlerinde ortaya çıkabilmektedir. Geliştirilen TocTou zafiyeti tespit etme yapısı çizelge 4.4’ de verilmiştir.

Çizelge 4.4. Toctou zafiyeti tespit etme sistemi çalışma altyapısı

No	TocTou Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Satırda Directory.File kullanıldıysa ve aynı satırda kullanımı gerçekleşmediyse
3	Toctou sayısı artırılır.
4	Alt satırlarda new fileinfo veya file.open komutu bulunana kadar Toctou sayısı artırılır.
5	Toctou sayısı 5 ve üzerindeyse muhtemel toctou zafiyetinin var olduğu kontrol edilir.

Güvensiz kod, Asp.Net teknolojisinde geliştirilen projelerde çalıştırılan kodlar ortak dil çalışma zamanı üzerinde derlenerek çalıştırılır. Bu sayede hafıza taşması gibi problemlerle ilgili önlemler alınmış olunur. Güvensiz kod, ortak dil çalışma zamanı (CLR) tarafından çalıştırılmayan ve kontrol edilmeyen kod bloklarıdır. Hafıza yönetimi geliştiriciye bırakılmıştır. Geliştirilen uygulamada satır satır kontrol edilen kod blokları arasında “unsafe” teriminin tespit edilmesi, bu kod satırının güvensiz kodlamaya sebep olduğunu ve test edilen projenin risk altında olduğunu göstermektedir. Güvensiz kodlama hafıza taşması veya hafıza zayıflığı zafiyetlerine neden olabildiği için zafiyet risk kategorisinin en üst basamaklarında yer almaktadır.

Xpath, XML dökümanı içerisinde bulunan veriyi sorgulamaya yarayan dildir. Web uygulamalarında veriler veri tabanı sunucusunda çeşitli şekillerde depolanabilir. Sunucudaki veri depolama formatı ilişkisel veritabanı yönetim

sistemi (RDBMS), LDAP veya XML gibi formatlarda olabilmektedir. Saldırı, saldırganın istemci tarafında kullanıcının veri girmesine izin verilen alanlardan girilen ya da sunucuya HTTP veya DNS istekleri aracılığıyla gönderilen, XPath sorgularıyla veritabanında yetkisiz veriye ulaşması sonucu oluşan saldırı şeklidir. Kaynak kod analiz yazılımında xpath enjeksiyonu oluşturabilecek satırları bulmak için çizelge 4.5’ de verilen aşamalar uygulanmaktadır.

Çizelge 4.5. Xpath enjeksiyonu tespit etme sistemi çalışma altyapısı

No	Xpath Enjeksiyonu Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Satırda xpathexpression ifadesi aranır.
3	Xpathexpression ifadesinin olduğu satırda temizleme işlemi gerçekleştirilip gerçekleştirilmediği kontrol edilir.
4	Gerçekleştirilmediyse Compile ifadesinin içerisindeki parametreler ayıklanır.
5	Parametreler Girdi ,Request nesnesi veya parametre mi kontrol edilir.
6	Girdi veya Request nesnesi ise o satırda Temizleme işleminin yapıp yapılmadığı kontrol edilir.
7	Yapılmadıysa Xpath Enjeksiyon satırının var olduğundan bahsedilebilir.
8	Parametre normal bir değişken ise üst satırlarda Temizleme işleminden geçip geçmediği kontrol edilir.
9	Hiçbir temizleme işleminden geçmediyse bu kod satırı Xpath zafiyeti barındırmaktadır denilebilir.

Ldap Enjeksiyonu, sql ve xpath enjeksiyonlarında olduğu gibi sorgulama satırlarlarına girdi verilerek yapılabilen bir saldırı türüdür. LDAP saldırılarında diğer saldırı tiplerinde olduğu gibi web uygulamalarının girdi alanları kontrol edilerek zafiyetin oluşabileceği yerler tespit edilir ve o satırlardaki önerilen değişiklikler geliştiriciye önerilir. Çizelge 4.6’ da LDAP Enjeksiyonu zafiyeti barındıran kod satırlarını tespit edebilmek için geliştirilen sistemin çalışma yapısı gösterilmiştir.

Çizelge 4.6. Ldap enjeksiyonu tespit etme sistemi çalışma altyapısı

No	Ldap Enjeksiyonu Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Satırda ldap sorgulama dilinde bulunan kodlar aranır.

Çizelge 4.6. Ldap enjeksiyonu tespit etme sistemi çalışma altyapısı (Devam)

3	Satırda Objectclass, cn, objectcategory tespit edildikten sonra satır temizleme işleminden geçirilmiş mi?
4	Gerçekleştirilmediyse satırdaki parametreler tespit edilir.
5	Parametreler Girdi ,Request nesnesi veya parametre mi kontrol edilir.
6	Girdi veya Request nesnesi ise o satırda Temizleme işleminin yapılıp yapılmadığı kontrol edilir.
7	Yapılmadıysa Ldap Enjeksiyon satırının var olduğundan bahsedilebilir.
8	Parametre normal bir değişken ise üst satırlarda Temizleme işleminden geçip geçmediği kontrol edilir.
9	Hiçbir temizleme işleminden geçmediyse bu kod satırı Ldap Enjeksiyonu barındırmaktadır denilebilir.

Header Enjeksiyonu, uygulama sunucusu ve oturum tanımlayıcısı arasına girerek web site veya uygulama üzerinde yetkiye sahip olabilen bir zafiyettir. Uygulamalarda, kullanıcı girdisinin http header a yazılma imkanı verildiği zamanlarda bu güvenlik açığı ortaya çıkabilmektedir. Oturum sabitleme, XSS gibi güvenlik açıklarına neden olabilmektedir. Asp.Net uygulamalarında header enjeksiyonunu tespit edebilmek için geliştirilen analiz aracında en uygun yöntem Response.Cookies bulunan satırları tespit etmektir. Geliştirilen analiz aracında tespit edilen satıra temizleme işlemi uygulanmış ise kod satırı güvenli olarak belirlenebilmektedir. Temizleme işlemi gerçekleşmediyse Response.Cookies ifadesinin almış olduğu parametreler tespit edilir. Tespit edilen parametreler Girdi nesnesi, Request nesnesi veya normal bir parametre olabilmektedir. Parametreler alt ve üst satırlarda kontrol edilerek temizleme işleminden geçirilip geçirilmediği belirlenerek kullanıcıya Header enjeksiyonu barındıran satırlar ile ilgili bilgi verilir.

Asp.Net Config Dosyası web uygulamalarındaki sunucu, veritabanı ve uygulama seviyesindeki ayarlamaların bulunduğu dosyadır. Config dosyası manipüle edilerek veya yanlış ayarlamalar yapılarak web uygulamalarında güvenlik zafiyetlerine sebep olmaktadır. Analiz aracında web config dosyası içerisinde satır satır okunan kod blokları içerisinde "<customerrors" satırı aranır. Mode ayarı "off" olarak ayarlanmışsa bu ayar kullanıcılara web sitesinin ayarlarıyla ilgili gizli bilgiler verebileceği için riskli satır olarak işaretlenmektedir. Debug

ayarlaması “true” olarak yapılmışsa hata ayıklama bilgilerini kullanıcılarla paylaşabilecektir. Böyle bir satır tespit edilmesi de riskli satır ve kodlama olarak işaretlenmektedir.

Temp klasörü, web uygulamalarında ayar dosyaları ve sabit web klasörlerinin sunucuda barındığı yer olarak bilinmektedir. Temp dizinindeki dosyaların manipüle edilmesi site hakkındaki ayarlamaların alınmasına yani race condition zafiyetine sebep olabilmektedir. Geliştirilen analiz aracında satır satır okunan veriler arasında file.open satırı aranır. Bu fonksiyon asp.net uygulamalarında dosya okumayı ve sonrasında dosya üzerinde işlemler yapmayı sağlamaktadır. File.open fonksiyonu parametre almaktadır. Parametrelerden önce “temp” cümleceği o satırda aranır. Eğer böyle bir ifade var ise güvenlik risk seviyesi en başından riskli kategorisinde değerlendirilir. Web kullanıcılarının doğrudan etkisi olup olmadığını ve zafiyet derecesini tespit etmek amacıyla kullanılan parametreler tespit edilir. Parametrelerin girdi parametresi, request nesnesi veya normal parametre olup olmadığı kontrol edilir ve güvenlik açıklığı derecelendirilir.

Yönlendirme hataları, doğrulanmayan yönlendirme olarak da bilinmektedir. Doğrulanmamış yönlendirmeler ile mevcut uygulama parametreleri değiştirilerek uygulamanın farklı içeriklere yönlendirilmesi sağlanılabilmektedir. Zararlı yazılım bulaştırma veya sosyal mühendislik yöntemleri kullanılarak çeşitli saldırılar gerçekleştirilebilir. Temelde kullanıcı girdilerinin kontrolsüz kullanımından kaynaklanmaktadır. Geliştirilen analiz aracında yönlendirme hatalarını engellemek için asp.net teknolojisinde kullanılan Response.Redirect(“”) fonksiyonu kontrol edilmektedir. Satır satır incelenen kodların arasında Response.Redirect fonksiyonu bulunduktan sonra fonksiyon içerisinde http ile başlayan bir kod bloğu aranır. Eğer böyle bir satır tespit edilirse güvenlik seviyesi düşük bir siteye yönlendirildiği ve zafiyet barındırabileceği söylenebilecektir. http bloğu yok ise fonksiyonun aldığı parametreler tespit edilmey çalışılır. Tespit edilen parametreler girdi parametresi, request nesnesi veya normal bir parametre olabilmektedir. Güvenlik risk derecelendirilmesi ile parametre türüne göre parametre bulunan satırın risk kategorisi belirlenir.

Log Kayıt kontrolü, asp.net uygulamalarında erişim bilgilerinin, tarih, saat, ip bazlı kayıt altına alınmasını sağlamaktadır. Bu kayıtlar yapılırken kullanılan fonksiyonlara kullanıcıların doğrudan erişim sağlayabileceği şekilde yapılan kodlama uygulamalar üzerinde zafiyetlere neden olabilmektedir. Geliştirilen uygulamada log alma sırasında oluşabilecek zafiyetleri tespit etmek için geliştirilen çalışma yapısı Çizelge 4.7’ de verilmiştir.

Çizelge 4.7. Log hatası tespit etme sistemi çalışma altyapısı

No	Log Kontrolü Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Satırda Logerror, Logger, Logging, System.Diagnostic aranır
3	Eğer 2. Satırdaki ifadeler geçiyorsa bu satırda password kelimesi log kelimesinden sonra geçiyorsa bu satırda şifre kaydediliyor denilebilmektedir.
4	Temizleme fonksiyonundan geçip geçmeme durumuna bakılarak risk derecelendirmesi yapılmaktadır.
4	Eğer taranan satırda 2. Satırdaki ifadeler geçiyor ve parametre ifadeleri bulunuyorsa
5	Parametreler Girdi ,Request nesnesi veya parametre mi kontrol edilir.
6	Girdi veya Request nesnesi ise o satırda Temizleme işleminin yapıp yapılmadığı kontrol edilir.
7	Yapılmadıysa Log Zafiyeti satırının var olduğundan bahsedilebilir.
8	Parametre normal bir değişken ise üst satırlarda Temizleme işleminden geçip geçmediği kontrol edilir.
9	Hiçbir temizleme işleminden geçmediyse bu kod satırı Log Hatası barındırmaktadır denilebilir.

Serileştirme probleminde web uygulamaları serileştirme veya geri getirme işlemleri gerçekleştirdiğinde nesnelerin kontrol edilmeden dosya sistemine yazılmasına neden olmaktadır. Serialization, nesnenin bir kopyasını kullanarak o kopyayı kolayca ulaşılabilir bir forma çevirme ve nesnenin o anki durumunu koruma işlemidir. Serialize yapılan nesneyi orjinal haline getirme işlemine ise Deserialization denir. Geliştirilen uygulamada serileştirme problemini tespit etmek için geliştirilen çalışma alt yapısı Çizelge 4.8’ de verilmiştir.

Çizelge 4.8. Serileştirme problemi tespit etme sistemi çalışma altyapısı

No	Serileştirme Problemi Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Satırda seriazible bulunmuş ise serializable=true olarak ayarlanır.
3	Serializable=true ise ve sonraki satırlarda Deserializable veya Readobject bulunuyorsa güvensiz kod satırı vardır
4	Serializable=false ve satır Runtime.Serilization içeriyorsa serializable=true olarak ayarlanır.
5	Eğer kod satırı serializable true ve kod satırı [Serializable içeriyorsa Classserializable değeri true olarak ayarlanır.
6	Seriziable=true ve ClassSeriziable=False ve kod satırı [assembly: SecurityPermission(veya [SecurityPermissionAttribute(içeriyorsa serializable ve classserializable değerleri false olarak işaretlenir.
7	ClassSeriziable=true ve kod satırı public class içeriyorsa güvensiz serileştirme problemi vardır ve kod satırı riskli kategoride değerlendirilmektedir.

4.1.2.2. Php dilinde incelenen zafiyetler

Php ile geliştirilen uygulamalarda sık rastlanan zafiyetler ve zafiyet oluşturabilecek kodlama problemleri şu şekildedir;

- *Sql Enjeksiyonu*
- *Xss (Çapraz Betik Sorgulaması)*
- *Xpath Enjeksiyonu*
- *Ldap Enjeksiyonu*
- *Header Enjeksiyonu*
- *Kayıt (Log) Kontrolü*
- *Dosya Doğrulama*
- *Dosya Ekleme*
- *Sistem Dosyalarının Çalıştırılması*
- *Backtick Hatası*
- *Parse_Str fonksiyonun Hatalı Kullanımı*
- *Php.ini dosyasındaki Zafiyetler*

Php ile web programlama yapılırken sayfalar arası veri transferini yapan

- *\$_GET*
- *\$HTTP_GET_VARS*
- *\$_POST*
- *\$HTTP_POST_VARS*
- *\$_REQUEST*
- *\$_COOKIE*
- *\$_SESSION*
- *\$_SERVER*
- *\$HTTP_SERVER_VARS* nesneleri geliştirilen yazılımda kontrol edilmektedir.

Geliştirilen analiz anacında php web programlama dilinde geliştirilen kodları filtrelemek için .php, .php3, php.ini uzantılı dosyalar ayıklanır. Ayıklanan dosyalardaki inputlar tüm projede taranarak tespit edilir. Böylece zafiyet barındıran kod satırlarının tespitine başlanmış olunur.

Sql Enjeksiyonu saldırısının php programlama dilindeki tespiti farklı programlama dillerine göre değişiklik göstermemektedir. Asp.Net’ de sql enjeksiyonunu engellemek için geliştirilen sistemin çalışma altyapısı php içinde kullanılabilecek şekilde tasarlanmıştır. Çizelge 4.1’ de verilen çalışma sistemi php de meydana gelen sql enjeksiyonu zafiyetlerini tespit etmek için eksiksiz uygulanmıştır. Temizleme fonksiyonları olarak da mevcut temizleme fonksiyonlarına ek olarak php deki hazır fonksiyonlar kullanılmıştır.

Xss Saldırısını php dilinde tespit edebilmek için, asp.net uygulamasında kurgulanan çalışma yapısının çok dışına çıkmamıştır. Dilin kendine özgü fonksiyonları kullanılarak kurgulanan çalışma sistemi Çizelge 4.9’ da verilmiştir.

Çizelge 4.9. Php dilinde xss saldırısı tespit etme sistemi çalışma altyapısı

No	XSS Saldırısı Uygulama Adımları
1	Projedeki .php .php3 uzantılı sayfalarındaki Girdi ve Çıktı Kontrolleri taranır.
2	Okunan Satırlarda veritabanı sorgusu bulunan satırlar dışındaki satırlar taranır.
3	Satırda Temizleme işlemi varsa o satır atlanır.
4	Satırda Temizleme işlemi yoksa echo, print veya print_r kontrollerinin olup olmadığı kontrol edilir.
5	Varsa bu kontrollerin aldığı parametreler kontrol edilir.
6	Aldığı parametreler Girdi veya Request Nesnesi ise ve Temizleme işleminden geçirilmediyse XSS zafiyeti vardır
7	Parametre normal bir parametreyse üst satırlarda temizleme işleminden geçip geçmediği kontrol edilir.
8	Satırda InnerHtml varsa ve eşitliğin sol bölümünde ise aldığı değer kontrol edilir.
9	Sağ bölümde aldığı değer Request nesnesi veya Girdi mi, Parametre mi olup olmadığı kontrol edilir.
10	Request veya Girdi ise temizleme işleminden o satırda geçmiş mi o kontrol edilir. Temizleme işleminden geçmediyse XSS zafiyeti vardır.
11	Parametre ise üst satırlarda kontrolden geçmiş mi kontrol edilir. Temizleme işleminden geçmediyse XSS zafiyeti vardır.

Çizelge 4.9. Php dilinde xss saldırısı tespit etme sistemi çalışma altyapısı(Devam)

12	Satırda document.write fonksiyonu var ise ve parametre alıyorsa
13	bu kontrolün aldığı parametreler kontrol edilir.
14	Aldığı parametreler Girdi veya Request Nesnesi ise ve Temizleme işleminden geçirilmediyse XSS zafiyeti vardır
15	Parametre normal bir parametreyse üst satırlarda temizleme işleminden geçip geçmediği kontrol edilir.

Xpath enjeksiyonunu php programlama dilinde tespit edebilmek için geliştirilen analiz aracında uygulanan tespit etme aşamaları Çizelge 4.10' da verilmiştir.

Çizelge 4.10. Php dilinde xpath enjeksiyonu tespit etme sistemi çalışma altyapısı

No	Php Dilinde Xpath Enjeksiyonu Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Satırda xpath ifadesi aranır.
3	Xpath ifadesinin olduğu satırda temizleme işlemi gerçekleştirilip gerçekleştirilmediği kontrol edilir.
4	Gerçekleştirilmediyse xpath ifadesinin içerisindeki parametreler ayıklanır.
5	Parametreler Girdi ,Request nesnesi veya parametre mi kontrol edilir.
6	Girdi veya Request nesnesi ise o satırda Temizleme işleminin yapıp yapılmadığı kontrol edilir.
7	Yapılmadıysa Xpath Enjeksiyon satırının var olduğundan bahsedilebilir.
8	Parametre normal bir değişken ise üst satırlarda Temizleme işleminden geçip geçmediği kontrol edilir.
9	Hiçbir temizleme işleminden geçmediyse bu kod satırı Xpath zafiyeti barındırmaktadır denilebilir.

Ldap Enjeksiyonunu php dilinde geliştirilen sayfalarda tespit edebilmek için asp.net teknolojisi için geliştirilen çalışma yapısı uygulanmaktadır. Analiz aracının php bölümünde ldap enjeksiyonunu tespit edebilmek için çizelge 4.6' daki adımlar uygulanmaktadır. Uygulama adımlarında php diline özgü ldap fonksiyonları farklılık göstermektedir. Php dilinde "ldap_add, ldap_bind, ldap_close, ldap_compare, err" gibi fonksiyonlar kullanılmaktadır.

Header enjeksiyonunu php dilinde tespit edebilmek için analiz aracında parametre alan header() fonksiyonu incelenmiştir. Header fonksiyonundaki parametreler girdi, istek nesnesi veya normal parametre durumlarına göre risk kategorilerinde sınıflandırılmıştır.

Log kontrolü sırasında kullanıcıların manipülasyonuna neden olabilecek güvenlik açıklarını php dilinde tespit edebilmek için geliştirilen sistemin çalışma altyapısı Çizelge 4.11’ de verilmiştir.

Çizelge 4.11. Php dilinde log hatası tespit etme sistemi çalışma altyapısı

No	Php Dilinde Log Kontrolü Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Satırda AddLog ve ErrorLog aranır.
3	Eğer 2. Satırdaki ifadeler geçiyorsa bu satırda ve password kelimesi log kelimesinden sonra geçiyorsa bu satırda şifre kaydediliyor denilebilmektedir.
4	Temizleme fonksiyonundan geçip geçmeme durumuna bakılarak risk derecelendirmesi yapılmaktadır.
4	Eğer taranan satırda 2. Satırdaki ifadeler geçiyor ve parametre ifadeleri bulunuyorsa
5	Parametreler Girdi ,Request nesnesi veya parametre mi kontrol edilir.
6	Girdi veya Request nesnesi ise o satırda Temizleme işleminin yapıp yapılmadığı kontrol edilir.
7	Yapılmadıysa Log Zafiyeti satırının var olduğundan bahsedilebilir.
8	Parametre normal bir değişken ise üst satırlarda Temizleme işleminden geçip geçmediği kontrol edilir.
9	Hiçbir temizleme işleminden geçmediyse bu kod satırı Log Hatası barındırmaktadır denilebilir.

Dosya işlemlerinde php dilinde kullanılan fonksiyonlar kullanıcı manipülasyonuna neden olabilmektedir. Dosyaları doğrulayabilmek için php dilinde \$_FILES fonksiyonunu ilk aşamada parametre olarak kullanılıp kullanılmadığına bakılmaktadır. Parametre tespit edilirse sonraki satırlarda dosyanın MIME type tipi kontrol edilerek risk kategorisi tespit edilmeye çalışılır. Parametre olarak kullanılmadıysa \$_FILES() içerisinde kullanılan parametrelerin içerisinde “type” tipinin bulunup bulunmadığı kontrol edilir. Risk kategorisi belirlenir.

Dosya Sızdırma File Inclusion olarak bilinmektedir. Php dilinde sıkça rastlanan bu güvenlik açığı dikkat edilmediği durumlarda kullanıcılara ve sistemlere büyük zararlar verebilmektedir. Geliştirilen sistemde Dosya sızdırma işlemlerini php dilinde engelleyebilmek adına geliştirilen sistem çizelge 4.12’ de verilmiştir.

Çizelge 4.12. Php dilinde dosya sızıntısı tespit etme sistemi çalışma altyapısı

No	Php Dilinde Dosya Sızıntısı Uygulama Adımları
1	Dosyalardaki satırlar tek tek okunur.
2	Satırda include_once, require_once, file_get_contents, fopen, fwrite readfile fonksiyonları aranır
3	Eğer 2. Satırdaki ifadeler geçiyorsa parametreler ayıklanır.
4	Temizleme fonksiyonundan geçip geçmeme durumuna bakılarak risk derecelendirmesi yapılmaktadır.
4	Eğer taranan satırda 2. Satırdaki ifadeler geçiyor ve parametre ifadeleri bulunuyorsa
5	Parametreler Girdi ,Request nesnesi veya parametre mi kontrol edilir.
6	Girdi veya Request nesnesi ise o satırda Temizleme işleminin yapıp yapılmadığı kontrol edilir.
7	Yapılmadıysa file inclusion hatasının var olduğundan bahsedilebilir.
8	Parametre normal bir değişken ise üst satırlarda Temizleme işleminden geçip geçmediği kontrol edilir.
9	Hiçbir temizleme işleminden geçmediyse bu kod satırında file inclusion barındırmaktadır denilebilir.

Komut Enjeksiyonu zafiyeti web programlama dillerinde görülebilen en önemli güvenlik açıklarındandır. Asp.Net ve Java gibi dillerde uygulama çatılarıyla kapatılabilen bu zafiyet Php dilinde yaygın olarak görülmektedir. Php dilinde daha sık görülen bu zafiyeti giderebilmek için kullanıcıların dışardan müdahalede bulunabileceği girdi alanlarına karşı alınabilecek önlemlerin başında, girdilere doğrudan maruz kalabilecek ve sistem üzerinde çalışabilecek fonksiyonların temizlenmesi gelmektedir. Diğer zafiyetleri giderme yöntemlerinde olduğu gibi bu zafiyette de öncelikle uygulama kodları satır satır okunur. Okunan satırlar arasında “exec, shell_exec, proc_open, eval, system, popen, passthru, pcntl_exec, assert” gibi php dilinde sunucu üzerinde komut çalıştırabilen fonksiyonlar taranır. Bulunan kod blokları içerisinde parametreler tespit edilerek parametrenin türüne ve risk kategorisine göre zafiyet tespit edilmeye çalışılır.

BackTick zafiyeti php dilindeki kodların komut satırında çalıştırabilmesini ters tırnak işareti kullanmasıyla sağlamaktadır. Geliştirilen analiz aracı `` ters tırnak işaretlerini arayarak içerisindeki parametreleri çıkarmaktadır. Parametreler girdi parametresi, request nesnesi ve normal parametre olarak tespit edilir. Bulunan parametreler risk kategorisine göre analiz aracı tarafından sınıflandırılır.

Parse_Str fonksiyonun hatalı kullanımı php dilinde kullanıcıların kullanılan dizgelere doğrudan erişimine ve yanlış veri alışverişine neden olmaktadır. Geliştirilen yazılımda “parse_str()” fonksiyonu kontrol edilir ve aldığı parametreler ayıklanarak türüne göre risk seviyesi belirlenir.

Php.ini dosyası php programlama dillerinde veritabanı, uygulama, izin gibi ayarlamaların barındığı web sitesi ayar dosyasıdır. Php için önemli olan bazı ayarlamalar geliştirilen analiz aracında kontrol edilerek elde edilen sonuçlar arayüzde gösterilir. Satır satır okunan kod blokları arasında register_globals bloğu aranır. Bu ayarlama post veya get ile gönderilen değerlere kullanıcı adlarıyla ulaşıp ulaşılamayacağını belirtir. “Off” olarak ayarlandığında bu gibi değerlere kendi tanımladığı şekilde ulaşılamamaktadır. “On” olarak ayarlanırsa bu gibi değerlere kendi tanımladığı şekilde ulaşılabilir ve riskli kategorisinde değerlendirilmektedir. Safe_mode ayarı ile script eklemeleri üzerinde ayarlamalar yapılabilmektedir. Safe_mode ayarı “off” yapılarak script eklenmesine izin verilmekte ve riskli kod bloğu olarak işaretlenmektedir. magic_quotes_gpc ayarlamasıyla SQLI saldırılarının önü kesilebilmektedir. Ayarlama “On” yapılarak SQLI saldırılarının önü kesilebilmektedir. Tarama sırasında karşılaşılan “Off” değeri riskli kod satırı olarak raporlanmaktadır.

4.1.2.3. Java’ da tespit edilen zafiyetler

Java ile geliştirilen uygulamalarda sık rastlanan zafiyetler ve zafiyet oluşturabilecek kodlama problemleri şu şekildedir;

- *Sql Enjeksiyonu*
- *Servlet Injection*

- *Xss (Çapraz Betik Sorgulaması)*
- *Xpath Enjeksiyonu*
- *Ldap Enjeksiyonu*
- *Runtime Çalışma Hatası*
- *Url Güvenliği*
- *Race Condition Zafiyeti*
- *Güvensiz Temp Dosyaları*
- *İzinsiz Dosya Çalıştırma Hatası*
- *Sistem Dosyalarına Erişim*
- *Xml Bomb Hatası*

Java ile web programlama yapılırken sayfalar arası veri transferini yapan

- *getParameter*
- *getParameterValues*
- *getHeader*
- *getContextPath*
- *getQueryString*
- *getSession* nesneleri

ile veritabanına sorgu yöndermeye yarayan

- *prepareStatement*
- *executeQuery*
- *query*
- *queryForObject*
- *queryForList*
- *queryForInt*
- *queryForMap*
- *createNativeQuery*
- *createQuery* geliştirilen yazılımda kontrol edilmektedir.

Geliştirilen analiz anacında java web programlama dilinde geliştirilen kodları filtrelemek için .jsp, .java, .jspx, .web.xml, .config.xml, .xhtml, .jsf uzantılı dosyalar ayıklanır. Ayıklanan dosyalardaki inputlar tüm projede taranarak tespit edilir. Böylece zafiyet barındıran kod satırlarının tespitine başlanmış olunur.

Sql Enjeksiyonu saldırısının java programlama dilindeki tespiti farklı programlama dillerine göre değişiklik göstermemektedir. Asp.Net’ de sql enjeksiyonunu engellemek için geliştirilen sistemin çalışma altyapısı java içinde kullanılabilecek şekilde tasarlanmıştır. Çizelge 4.1’ de verilen çalışma sistemi java da meydana gelen sql enjeksiyonu zafiyetlerini tespit etmek için eksiksiz uygulanmıştır. Temizleme fonksiyonları olarak da mevcut temizleme fonksiyonlarına ek olarak java’ daki hazır fonksiyonlar kullanılmıştır.

Xss Saldırısını java dilinde tespit edebilmek için, Asp.net uygulamasında kurgulanan çalışma yapısının çok dışına çıkılmamıştır. Dilin kendine özgü fonksiyonları kullanılarak kurgulanan çalışma sistemi çizelge 4.13’ de verilmiştir.

Çizelge 4.13. Java dilinde xss saldırısı tespit etme sistemi çalışma altyapısı

No	Java da XSS Saldırısı Uygulama Adımları
1	Projedeki java sayfalarındaki Girdi ve Çıktı Kontrolleri taranır.
2	Okunan Satırlarda veritabanı sorgusu bulunan satırlar dışındaki satırlar taranır.
3	Satırda Temizleme işlemi varsa o satır atlanır.
4	Satırda Temizleme işlemi yoksa out.println kontrollerinin olup olmadığı kontrol edilir.
5	Varsa bu kontrollerin aldığı parametreler kontrol edilir.
6	Aldığı parametreler Girdi veya Request Nesnesi ise ve Temizleme işleminden geçirilmediyse XSS zafiyeti vardır
7	Parametre normal bir parametreyse üst satırlarda temizleme işleminden geçip geçmediği kontrol edilir.
8	Satırda <%= varsa aldığı değer kontrol edilir.
9	Sağ bölümde aldığı değer Request nesnesi veya Girdi mi, Parametre mi olup olmadığı kontrol edilir.
10	Request veya Girdi ise temizleme işleminden o satırda geçmiş mi o kontrol edilir. Temizleme işleminden geçmediyse XSS zafiyeti vardır.
11	Parametre ise üst satırlarda kontrolden geçmiş mi kontrol edilir. Temizleme işleminden geçmediyse XSS zafiyeti vardır.
12	Satırda document.write fonksiyonu var ise ve parametre alıyorsa
13	bu kontrolün aldığı parametreler kontrol edilir.
14	Aldığı parametreler Girdi veya Request Nesnesi ise ve Temizleme işleminden geçirilmediyse XSS zafiyeti vardır
15	Parametre normal bir parametreyse üst satırlarda temizleme işleminden geçip geçmediği kontrol edilir.
16	Satırda <c:out varsa escapexml değeri true olarak ayarlanmamış ise XSS zafiyeti vardır deilebilir.

Servletler, java dilinde kullanıcıdan aldığı verilere göre sonuçlar üreten Java sınıflarıdır. Bu veriler üzerinde gereken işlemler yapıldıktan sonra önceden tasarlanmış biçimlere uyarlanarak kullanıcıya geri döndürülür. Servlet’ de bulunan bir zafiyet tüm sistemi etkilemektedir. Geliştirilen analiz aracında satır satır okunan veriler arasında “extends HttpServlet” bloğu taranır. Blok bulunduğu taktirde sonraki satırlarda thread.sleep fonksiyonu aranır. Fonksiyonun bulunması servletlerde önerilmemektedir ve analiz aracında riskli kategoride değerlendirilmektedir.

Ldap Enjeksiyonunu java dilinde geliştirilen sayfalarda tespit edebilmek için asp.net teknolojisi için geliştirilen çalışma yapısı uygulanmaktadır. Analiz aracının java bölümünde ldap enjeksiyonunu tespit edebilmek için çizelge 4.6' daki adımlar uygulanmaktadır. Uygulama adımlarında php diline özgü ldap fonksiyonları farklılık göstermektedir. Java dilinde objectClass ve objectcategory gibi fonksiyonlar kullanılmaktadır.

Xpath enjeksiyonunu java programlama dilinde tespit edebilmek için çizelge 4.10' daki uygulama adımları uygulanır. Fakat Java diline özgü xpathexpression ifadesi kod satırında aranır. İfadenin olduğu satırda temizleme işleminin gerçekleşip gerçekleşmediği kontrol edilir. Gerçekleşmediyse “.Compile” fonksiyonunun içerisindeki parametreler ayıklanır. Ayıklanan parametreler türüne göre sınıflandırılır ve dosyadaki satırlar arasında arama gerçekleştirilir. Sınıflandırılan parametreler risk kategorilerine ayrılır.

Çalışma Zamanı komutu işletim sistemi komut satırında program parçacığı çalıştırmayı sağlar. Fakat komut satırında kod çalıştırmak riskli bir işlemdir. Analiz aracında getruntime().exec() komutu aranarak kullanıcıların müdahalede bulunup bulunamayacağı kontrol edilir. Parametreler ayıklanarak girdi veya istek nesnesi olup olmadığı kontrol edilir. Normal bir parametre ise tüm dosyada temizleme işleminden geçip geçmediği kontrol edilir. Parametreler sınıflandırılır ve risk kategorilerine ayrılır.

Adres kontrolü, farklı sitelere yönlendirme işlemlerinde bilinmeyen sitelerin güvensizliğine karşı, geliştiricilere bilgi verebilmesi amacıyla yapılmaktadır. Ayrıca yönlendirme işleminin kullanıcıların hakimeyitine bırakılması zafiyetlere neden olabilmektedir. Analiz aracında öncelikle yönlendirilen adreslerin https güvenliği kontrolü yapılır. Satır satır incelenen kod bloklarında http yönlendirilmesi var ise kaynak kod satırı güvensiz olarak işaretlenmektedir. Ayrıca inceleme sırasında URLConnection ve openConnection fonksiyonları aranarak fonksiyonların aldıkları parametreler tespit edilir. Parameterlerin

türlerine göre risk kategorileri tespit edilir ve kod bloklarındaki adres zafiyeti barındıran sınıflar tespit edilmiş olunur.

Race Condition zafiyeti, bir sistemin veya cihazın aynı kaynak üzerinden eş zamanlı olarak işlem yapmasıyla meydana gelir. Java da race condition zafiyetini engellemek için çizelge 4.14.' deki çalışma yapısı uygulanmaktadır.

Çizelge 4.14. Java dilinde race condition saldırısı tespit etme sistemi çalışma altyapısı

No	Java' da Race Condition Saldırısı Uygulama Adımları
1	Projedeki java sayfalarında bulunan Girdi ve Çıktı Kontrolleri taranır.
2	Okunan Satırlarda new thread fonksiyonu aranır.
3	Alt satırların tamamında synchronized ifadesi taranır.
4	Kullanılan satır var ise Race Condition engellenmiştir.
5	Syncronized ifadesi tüm satırlarda bulunamıyorsa Race Condition zafiyeti vardır.
6	New thread fonksiyonunun bulunduğu satırlardan sonra lock() fonksiyonu kullanılmışsa Race Condition engellenmiştir.
7	Tüm satırlarda unlock fonksiyonu bulunamıyorsa Race Condition zafiyeti vardır.

Temp Dosyalarını kullanarak oluşabilecek zafiyetleri java programlama dilinde tespit edebilmek için geliştirilen analiz aracında satır satır okunan veriler arasında new file satırı aranır. Bu fonksiyon java uygulamalarında dosya okumayı ve sonrasında dosya üzerinde işlemler yapmayı sağlamaktadır. New file fonksiyonu parametre almaktadır. Parametrelerden önce “temp” cümleceği o satırda aranır. Eğer böyle bir ifade var se güvenlik risk seviyesi en başından riskli kategorisinde değerlendirilir. Web kullanıcıların doğrudan etkisi olup olmadığını ve zafiyet derecesini tespit etmek amacıyla kullanılan parametreler tespit edilir. Parametrelerin girdi parametresi, request nesnesi veya normal parametre olup olmadığı kontrol edilir ve güvenlik açıklığı derecelendirilir.

İzinsiz Dosya Çalıştırma metodu ile normalde izin hakkı olmayan uygulamaların kritik işlemleri geçici süre ile gerçekleştirmelerini sağlanmaktadır. Javada kullanılan “AccessController.doPrivileged” metodu bu işlemi gerçekleştirmektedir. Analiz aracının javadaki kritik işlemlere izin verdiği alanlarda bu zafiyeti tespit edebilmesi için tüm satırlarda

“AccessController.doPrivileged” metodu izlenir. Metodun içerisindeki parametreler tespit edilir. Parametrelerin girdi nesnesi olup olmadığı kontrol edilir. Parametre bloğunun içerisinde new file metodu kullanılmışsa ise zafiyetin kategorisi riskli olarak belirlenmektedir. Blok içi satır sayısı 20 den fazla ise riskli, blok içi satır sayısı 10’ dan büyükse yarı riskli ve 10’ dan küçükse risk seviyesi standart olarak belirlenmektedir.

RequestDispatcher yontemi istek anında baska jsp ya da servlet sayfalarını yönlendirmek için kullanılmaktadır. Java Servlet basit bir parametre ile birlikte aldığı web sayfasını, nitelikleri ile birlikte hedef sayfaya ulaştıracaktır. Bir servlet’ ten diğer servlet’ e yönlendirme olarak bilinmektedir. Yöntemin kullanılmasıyla saldırgan uygun ortam bulduğunda sistem dosyalarını indirebilmektedir. Analiz aracı bu zafiyeti engelleyebilmek için satır satır taranan dosyalarda temizleme işleminden geçip geçmediği kontrol edilir. Temizleme işleminden geçmeyen satırlar için getRequestDispatcher fonksiyonu içerisindeki parametreler taranır. Parametreler girdi, istek ve normal parametre türlerine göre risk seviyesi kategorize edilir.

Xml bomb koruması xml dosyalarının genişletme saldırısına karşı bir yöntemdir. Java programlama dilinde bu zafiyeti engellemek için satır satır taranan kod bloklarında “import javax.xml.bind.JAXB” satırı aranır. Bu dll’ in kullanılmasıyla birlikte alt satırlarda XMLConstants.FEATURE_SECURE_PROCESSING değerinin false olarak belirlenmesi veya XMLInputFactory.SUPPORT_DTD değerinin false olarak belirlenmesi ile Xml Bomb çalıştırabilmektedir ve kod bloğu riskli bir satır olarak işaretlenmektedir.

4.1.3. Metrik analizi

Yazılım güvenliği test ve değerlendirme aracının metrik analizini yapabilmesi için analiz ayarları sekmesinde kullanıcılar analizini gerçekleştireceği projeyi gözet düğmesine tıklayarak yükleyebilmektedirler. Proje dilinde Java, asp.net ve php dilleri seçilebilir. Arayüzün sol bölümünde yüklenen proje ağaç yapısında gösterilmektedir. Herhangi bir dosya seçilerek dosya içeriği aynı arayüzde incelenebilmektedir. Analiz türü seçeneği ile tekli dosya veya tüm projenin metrik analizi yapılabilir.

Çalışma kapsamında geliştirilen metrik analizinin çalışma mantığında öncelikle kullanıcıdan alınan proje dosyalarının işlenmesi aşaması bulunmaktadır. Projelerin içerisindeki resim, video gibi analize gerek duyulmayan dosya tipleri çıkarılarak filtreleme gerçekleştirilir. Arındırılan dosyaların analizi için her bir programlama dilindeki dosyalar satır satır metrik türlerine göre incelenir. İnceleme sonucunda sonuçlar kategorilerine göre raporlanır.

4.1.3.1. Asp.net, Php ve Java dillerinde ölçülen metrik türleri

Yazılım dillerindeki kod satırları üzerinde anlamlı çıkarımlarda bulunmak ve yazılım kalitesi hakkında yorum yapabilmek için kod satırları üzerinde hesaplamalar yapılarak istatistiksel veriler elde edilmektedir. İstatistiksel hesaplamalar farklı programlama dillerinde benzer yöntemlerle yapılabilmektedir. Çalışma kapsamında geliştirilen analiz aracı, kod satırları üzerinde metrik analizi yapmaktadır. Metriklerin ölçüm sistemi ise geliştirilen yazılımda, kod satırlarını parçalara ayırarak analizi gerçekleştirmektedir.

Bu çalışmada Asp.Net, Php ve Java dillerinde ölçülen metrikler şu şekildedir;

- Satır Sayısı
- Kaynak Kod Satır Sayısı
- Boş Satır Sayısı
- Yorum Satır Sayısı
- Yorum Oranı
- Çevrimsel Karmaşıklık
- Metot Sayısı
- Bakım İndeksi
- Sınıf Başına Ağırlıklı Metot
- Halstead Metrikleri

Tüm dillerde programdaki satır sayısı bulunurken toplam satır sayısı hesaplanır. Kaynak kod satır sayısı ise yorum ve boş satır sayısı olmadan sayılmaktadır. Boş satır sayısı satırın hiçbir değer içermemesi, yorum satırı sayısının hesaplanması ise dillere göre farklılık göstermektedir. Asp.Net teknolojisinde ve Java dilinde yorum satırları “//” ile başlayabilirken açıklamalı yorum satırları ise /* */ ifadeleri arasına sıkıştırılmaktadır. Php dilindeki yorum satırları ise token_get_all fonksiyonu ile parçalara ayrılan kodlar arasında T_COMMENT T_DOC_COMMENT

içeren satırlar sayılarak hesaplanmaktadır. Yorum oranı ise Yorum Satır Sayısının, kaynak kod satır sayısına bölümüyle hesaplanmaktadır.

Çevrimsel Karmaşıklık: Thomas J. McCabe tarafından ortaya konan bu metrik bir metodun içindeki algoritmanın karmaşıklığını ölçmek için kullanılır (McCabe, 1976). Yazılımlardaki karmaşıklık metotların içindeki koşullu ifadelerle hesaplanabilmektedir (Garg, 2014). Çevrimsel karmaşıklık eşitlik 4.1.' deki formül yardımıyla hesaplanabilmektedir (Watson ve McCabe, 1996).

$$CC = E - N + 2 \quad (4.1)$$

Eşitlik 4.1.' deki E değeri algoritmadaki yolları, N ise algoritmadaki düğümleri ifade etmektedir. Geliştirilen analiz aracının metrik hesaplamaları bölümünde çevrimsel karmaşıklık değeri asp.net ve java dillerinde "if", "elseif", "case", "break", "continue", "do", "finally", "for", "switch", "while" ifadeleri arayarak tespit edilen nesneleri sayılarak elde edilmektedir. Phpde ise parçalara ayrılan kod blokları içerisinde token değeri olarak "T_ELSEIF", "T_FOR", "T_FOREACH", "T_WHILE", "T_CASE", "T_CATCH", "T_BOOLEAN_AND", "T_LOGICAL_AND", "T_BOOLEAN_OR", "T_LOGICAL_OR" ifadeleri aranarak çevrimsel karmaşıklık değeri bulunmaktadır. Sınıf başına ve metod başına çevrimsel karmaşıklık değeri de sistem içerisinde hesaplanmaktadır.

Sınıf başına düşen ağırlıklı metod sayısı (WMC): Döngüsel karmaşıklık değerleri toplamının sınıf sayısına bölünmesi hesaplanır. WMC, sınıfın geliştirilmesi ve bakımı için harcanması gereken zamanı ve eforu göstermektedir (Buzluca, 2012). WMC değeri eşitlik 4.2.' deki formül yardımıyla hesaplanabilmektedir (Buzluca, 2013).

$$WMC = \frac{CC}{LOC} * \frac{LOC}{Method} * \frac{NOM}{Class} \quad (4.2)$$

Eşitlik 4.2.' deki CC değeri Çevrimsel Karmaşıklık, LOC değeri satır sayısını, Method değeri Sınıftaki metod sayısını, NOM ifadesi fonksiyon Sayısını, Class ise

sınıf sayısını ifade etmektedir. Asp.Net ve Java’ da Eşitlik 4.2.’ deki formül kullanılarak wmc değeri dosya bazında hesaplanmaktadır.

Bakım İndexi: Sınıf üyeleri seviyesinde kod bakımının kolaylığını gösterir. Değerin yüksek olması programın sürdürülebilirlik seviyesinin yüksek olduğu anlamına gelmektedir. Bakım endexi, analiz aracında eşitlik 4.3.’ deki gibi hesaplanmaktadır (Conor, 2007).

$$MIndex = MAX(0, (172 - 5.2 * \ln(HV) - 0.23 * (CC) - 16.2 * \ln(LOC)) * \frac{100}{171}) \quad (4.3)$$

Eşitlik 4.3.’ deki HV değeri Halstead Volume, CC değeri çevrimsel karmaşıklık, LOC değeri kod satırı Sayısını ifade etmektedir. Asp.Net’ te hesaplanan bakım indexi değeri için öncelikle Halstead değeri hesaplanır. Hesaplanacak dosyanın operatör ve operand sayıları toplanarak büyüklük hesaplanır. Hesaplanan büyüklük değerinin logaritması alınarak Program uzunluğu ile çarpılır. Program uzunluğu operatör ve operand değerlerinin toplamıyla hesaplanır. Eşitlik 4.3.’ deki formül kullanılarak bakım index değeri hesaplanır.

NOM: Sınıftaki fonksiyonların toplamı alınarak hesaplanır. Java, Asp.Net ve Php dilinde parametre alan ve almayan fonksiyonlar kod satırlarının içerisinde çıkarılarak hesaplanır.

Analiz aracına yüklenen projelerin hesaplanan metrikleri dosya bazında kullanıcılara geliştirilen dile göre rapor verebilmektedir.

4.1.4. Stil Denetimi

Stil denetimi geliştiricilerin uygulamalarını belirli standartlara göre geliştirmesi temeline dayanmaktadır. Geliştirilen kodun sürdürülebilir ve güvenilir olmasından çok, yüksek performanslı olması göz önünde tutulduğu yazılımlarda uzun vadede verimlilik ve performans, güvenilirlik ve bakımdan sonra gelmektedir. Yazılan kodun güvenilir ve sürdürülebilir olmaması, sorunların tespiti ve uygulama akışının anlaşılabilmesi için çok fazla zaman harcanmasına

neden olmaktadır. Geliştirilen yazılımın odak noktasının hızlı yazılım üretmekten ziyade, yazılan kodun bakımının kolay olması, büyük yazılım firmalarının ön planda tuttuğu bir durumdur. Geliştirilen kodların belirli standartlarda olabilmesi için asp.net, php ve java standartlaşmış ve kabul edilmiş stil denetim kuralları kullanılarak stil denetim gerçekleştirilmiştir.

Yazılım güvenliği test ve değerlendirme aracının stil denetimi yapabilmesi için analiz ayarları sekmesinde kullanıcılar analizini gerçekleştireceği projeyi gözet düğmesine tıklayarak yükleyebilmektedirler. Proje dilinde Java, asp.net ve php dilleri seçilebilir. Arayüzün sol bölümünde yüklenen proje ağaç yapısında gösterilmektedir. Herhangi bir dosya seçilerek dosya içeriği aynı arayüzde incelenebilmektedir. Analiz türü seçeneği ile tekli dosya veya tüm projenin stil denetimi yapılabilir.

Çalışma kapsamında geliştirilen stil denetiminin çalışma mantığında öncelikle kullanıcıdan alınan proje dosyalarının işlenmesi aşaması bulunmaktadır. Projelerin içerisindeki resim, video gibi analize gerek duyulmayan dosya tipleri çıkarılarak filtreleme gerçekleştirilir. Arındırılan dosyaların analizi için her bir programlama dilindeki dosyalar satır satır stil denetim kurallarına göre incelenir. İnceleme sonucunda sonuçlar denetim kurallarına göre raporlanır.

4.1.4.1. Asp.net, Php ve Java dillerinde uygulanan stil denetimleri

Çalışma kapsamında geliştirilen analiz aracı kod satırları üzerinde stil denetimi yapmaktadır. Geliştirilen yazılımda stil denetimleri, kod satırları parçalara ayrılarak yapılmıştır.

Asp.Net ile geliştirilen uygulamalarda stil denetimi sınıf, değişken, fonksiyon, namespace, interface, enumeration, const ve eventhandler olmak üzere 8 kategoride yapılmaktadır.

Denetim işlemine başlamadan önce kurallara özgü sabit ifadeler tespit edilmiştir. Asp.Net’ de istenmeyen özel ifadeler şu şekildedir; “abstract, while, lock, stackalloc, else , static , namespace , volatile , delegate , internal , short , void , do ,

is , sizeof , continue , in , return , using , decimal , sbyte , virtual , default , interface , sealed , checked , goto , public , unchecked , class , if , readonly , unsafe , const , implicit , ref , event , try , as , case , float , params , typeof , catch , for , private , foreach , protected , extern , new , object , this , bool , fixed , override , base , throw , true , byte , out , finally , break , operator , false , switch , explicit , null , struct , int , string , int [] , string [] , char , char [] , bool , byte , float , params , typeof , uint , ulong , ushort , decimal , sbyte , short , double , long , enum”.

Tüm denetim kategorilerinde öncelikle düzenli ifadeler kullanılarak birer birer okunan dosyalardaki “sınıf, değişken, fonksiyon, namespace, interface, enumeration, const ve eventhandler” isimleri tespit edilir. İsimlerde özel karakter kullanılıp kullanılmadığı, isimlerin _ veya - kullanılıp kullanılmadığı ve Türkçe karakter kullanılma durumları raporlanır. Kullanılma durumları risk kategorisine girmektedir.

Ek olarak sınıf isimlerinin içerisinde sınıf, sinif, class değerlerinin geçip geçmediği de analiz kapsamında kontrol edilir. Sınıf isimlerinde Camel Casing kuralının uygulanmış olması denetlenmiştir. Değişken isimlerinde get, set, al kullanılıp kullanılmadığı kontrol edilir. Fonksiyon, namespace, interface, enumeration, const isimlerinde Pascal Casing kuralının uygulanmış olması denetlenmiştir. Interface isimlerinin I harfiyle başlamış olması denetlenmiştir. Enumeration isimlendirmelerinin içerisinde “enum” ifadesinin geçmesi aranmıştır. Eventhandler isimlerinde “on” ifadesinin geçmesi istenmeyen kullanım şekli olarak belirlenmiştir.

Php ile geliştirilen uygulamalarda stil denetimi başlangıç tagları, sınıf, const, fonksiyon, namespace, maximum satır sayısı, süslü parantez, boolean değerler, parametre, use kullanım, Implements, fonksiyon parantezleri, elseif ifadeleri olmak üzere 13 kategoride yapılmaktadır.

Php dosyalarının başlangıç tagı <?php veya <?= ile başlayıp başlamadığı kontrol edilmiştir. Dosyanın son satırı ?> ile bitip bitmediği raporlanmıştır. Dosyalardaki sınıf isimleri tespit edilerek her kelimenin ilk harfinin büyük olma(Pascal Casing) kuralının uygulanıp uygulanmadığı aranmıştır. Sabit adlarında tüm karakterlerin

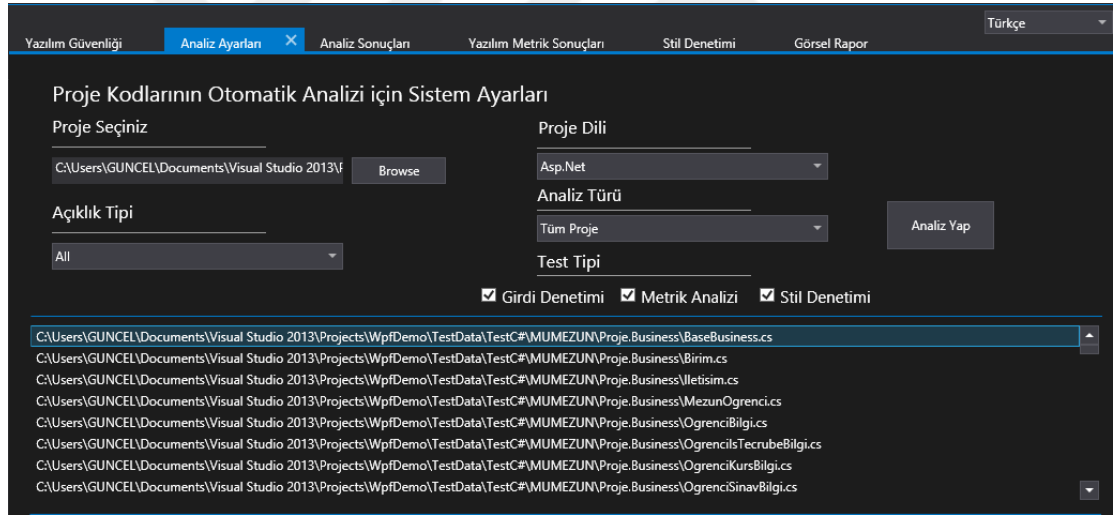
büyük harf olması denetlenmiştir. Büyük harf değilse veya _ ifadesi varsa istenmeyen satır olarak işaretlenmektedir. Fonksiyon isimlerinde ilk sözcüğün küçük harfle diğer sözcüklerin büyük harfle başlama kuralının (Camel Casing) uygulanıp uygulanmadığı denetlenmiştir. Sınıf tanımlarının olduğu satırlarda isim uzayı (namespace) ifadesinin sınıf tanımlarından önce olması gerektiği denetlenmiştir. Php dosyalarındaki maximum satır sayısının 80 den küçük olduğu durumlarda normal, 80 ve 120 arasında yarı riskli ve 120 den büyük olan yerlerde riskli olarak işaretlenmektedir. Php dosyalarında sınıf ve fonksiyon tanımlarının olduğu satırlarda süslü parantez kullanılmış ise riskli satır olarak işaretlenmektedir. False, true ve null ifadelerinin tüm harflerinin büyük harfle yazılması “psr” standartlarına göre analiz aracında aranan özelliktir. Php deki parametrelerin tüm harflerinin küçük harfle yazılmış olması gerektiği analiz aracında aranan özelliktir. Php kapanış tagından sonra boş satır bırakılmaması riskli bir satır olarak işaretlenmektedir. Namespace kullanımından sonra sayfanın sonlanması veya namespace kullanımından sonra boş satır bırakılmaması riskli kod olarak belirlenmektedir. Use kullanımından önce namespace kullanılmaması veya use satırından sonra boş satır bırakılmaması riskli kod olarak işaretlenmektedir. “Extends” veya “implements” tanımlarının sınıf tanımıyla aynı satırda yapılmaması riskli kod satırı olarak belirlenmektedir. Fonksiyonların kullanıldıkları yerde açılış parantezlerinden sonra veya kapanış parantezden önce boşluk bırakılması riskli kod olarak belirlenmiştir. Else ile başlayan kod satırında süslü parantezin aynı satırda olması aranmaktadır. Else if ifadesinin bitişik yazılması denetlenmektedir. Geliştirilen analiz aracında yukarıdaki paragrafta listelenen tüm özellikler uygulanmıştır.

Java ile geliştirilen uygulamalarda stil denetimi “paket, sınıf, arayüz, enum, değişken, collection, boolean, public static final” olmak üzere 8 kategoride yapılmıştır.

Paket denetiminde dosyadaki paket ismi çıkarılır. Paket isimlerindeki domain isimlerinin tersten yazılmış olması ve tüm karakterlerin küçük harften oluşuyor olması aranmıştır. “Sınıf, Arayüz, Enum, Değişken, Collection, Boolean” değişken veya fonksiyon denetiminde tespit edilen isimde özel karakter kullanılıp

kullanılmadığı ve _- işaretlerinin kullanılıp kullanılmadığı aranmaktadır. İsimde Türkçe karakter kullanıldı ise zararlı kod satırı olarak işaretlenmektedir. Ayrıca isim tanımında Pascal Casing istenmektedir. Enum tanımında “enum” ifadesinin geçmesi aranılan bir özelliktir. Değişken adının içerisinde get, set, al kullanılması riskli bir ifadedir. Collection tanımında “s” takısının olması standartlar arasında kod satırlarında aranmıştır. Boolean değişkenlerde “is” önekinin olması aranmıştır. Public static final değişkenler _ veya - kullanmalıdırlar. Ayrıca tüm harflerinin büyük olması da java kod standartlarında istenen bir özelliktir. Bu paragrafta istenen tüm özellikler geliştirilen analiz aracına eklenmiştir.

Geliştirilen kaynak kod analiz aracında kullanıcıların web uygulamalarındaki güvenlik taraması 3 aşamada yapılabilmektedir. Girdi denetimi, metrik analizi ve stil denetimi için analiz ayarları sekmesi kullanılabilmektedir. Girdi denetimine ait ekran görüntüsü şekil 4.3’ de verilmiştir.



Şekil 4.3. Girdi denetimi arayüzü

Analiz ayarları sekmesinde yüklenen projeye ait arındırılmış dosyalar alt bölümde listelenmiştir. Girdi denetimi için proje dili seçimine göre açıklık tipleri seçilerek çeşitli analizler yapılabilmektedir. Şekil 4.4’ de açıklık tiplerine ait seçim arayüzü gösterilmiştir.

Açıklık Tipi

All

All

Sql Injection

Cross Site Scripting (XSS)

Xpath Injection

Ldap Injection

Header Injection

Input Validation

Secure String Storage

Log Checking

TocTou

Serilization Problem

Redirect Checking

UnSecure Temp Files

Unsafe Code

Check Executable

Check WebConfig

Proje Dili

Asp.Net

Asp.Net

Php

Java

Şekil 4.4. Girdi denetimi açıklık tipi seçim arayüzü

Açıklık tipleri seçilen proje diline göre değişmektedir. Kullanıcılar istenildiği takdirde tüm açıklıkları arayabileceği gibi tek tek seçim yapılarak özel analiz de yapılabilmektedir. Metrik analizi ve stil denetimi için ayrı analiz ayarları bulunmamaktadır. Kullanıcılar sadece analiz türü ve proje dilini seçerek analizleri yapabilmektedirler, Şekil 4.5' de girdi denetimi, metrik analizi ve stil denetiminin nasıl yapılabileceği gösterilmiştir.

Analiz Türü

Tüm Proje

Tüm Proje

Seçili Dosya

☒ Girdi Denetimi ☒ Metrik Analizi ☒ Stil Denetimi

Analiz Yap

Şekil 4.5. Analiz türü seçim arayüzü

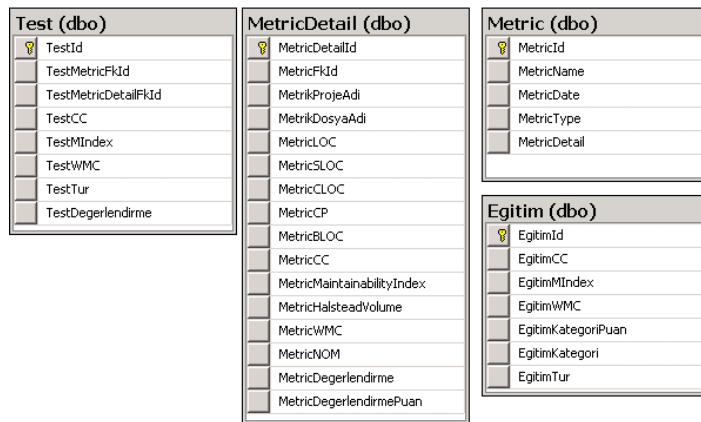
Yüklenen projedeki seçilen dosya analiz edilebileceği gibi tüm proje de analiz tiplerine göre denetlenebilmektedir.

4.2. Güvenlik Seviyesi Belirleme

Güvenlik seviyesi belirleme yazılımı Asp.Net teknolojisi kullanılarak Visual Studio 2013 ortamında web uygulaması olarak geliştirilmiştir. Web uygulaması olarak geliştirilen güvenlik yazılımı, programların kod kalitesine göre güvenlik seviyesini tespit etmeyi, aynı zamanda bu alanda kendini geliştirmek isteyenlere ve geliştiricilere güvenlik bilincini kazandırmayı amaçlamıştır. Online analiz aracı bilgi güvenliği eğitimi veren yerlerde ve lisans veya lisansüstü derslerde de kullanılabilir. İnternet erişimi olan her yerden erişimi sayesinde bilgisayara herhangi bir kurulum gerektirmemektedir. Sistemi kullanmak için üye olmak ve sonrasında kullanıcı hesap bilgileriyle giriş yapmak yeterlidir. Kullanıcılar yaptıkları analizlerin geriye dönük bir şekilde raporlarını alabilmektedirler. Geliştirilen online güvenlik aracı projeleri metrik türlerine göre incelerken, çalışmanın ikinci bölümünde metriklerin literatürde kabul görmüş eşik değerlerini kullanarak güvenlik derecelendirmesi gerçekleştirilmektedir.

4.2.1. Online güvenlik yazılımı veritabanı

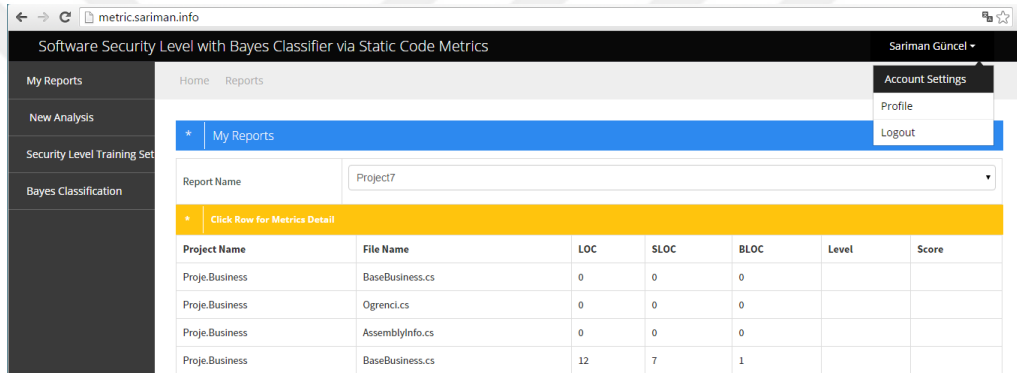
Geliştirilen yazılımın kullanıcılara geriye dönük raporlama verebilmesi amacıyla, eğitim ve test verilerinin kaydedilebilmesi için oluşturulan veritabanı diyagramı şekil 4.6' da verilmiştir.



Şekil 4.6. Güvenlik seviyesi belirleme yazılımı veritabanı diyagramı

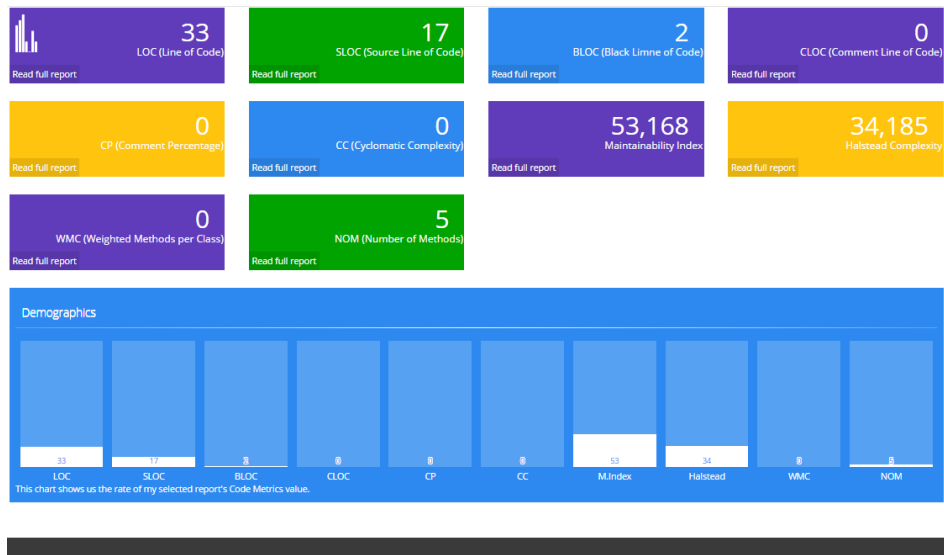
4.2.2. Güvenlik seviyesi belirleme yazılımı

Yazılımlar üzerinden rakamlarla konuşabilmeyi sağlayan kod metrikleri geliştirilen uygulamaların kod kalitesi hakkında bilgiler vermektedir. Kod kalitesinin yazılımların güvenliği üzerindeki etkisi hesaba katılarak güvenlik yaklaşımını kaynak kod kalitesiyle birleştiren bu çalışma sonunda yazılım projelerindeki dosyalar risk kategorisine göre sınıflandırılmaktadır. Online güvenlik yazılımı <http://metric.sariman.info> adresinde tüm kullanıcılar tarafından kullanılabilir. Şekil 4.7 ve Şekil 4.8’ de geliştirilen sitenin genel bir görünümü verilmiştir.



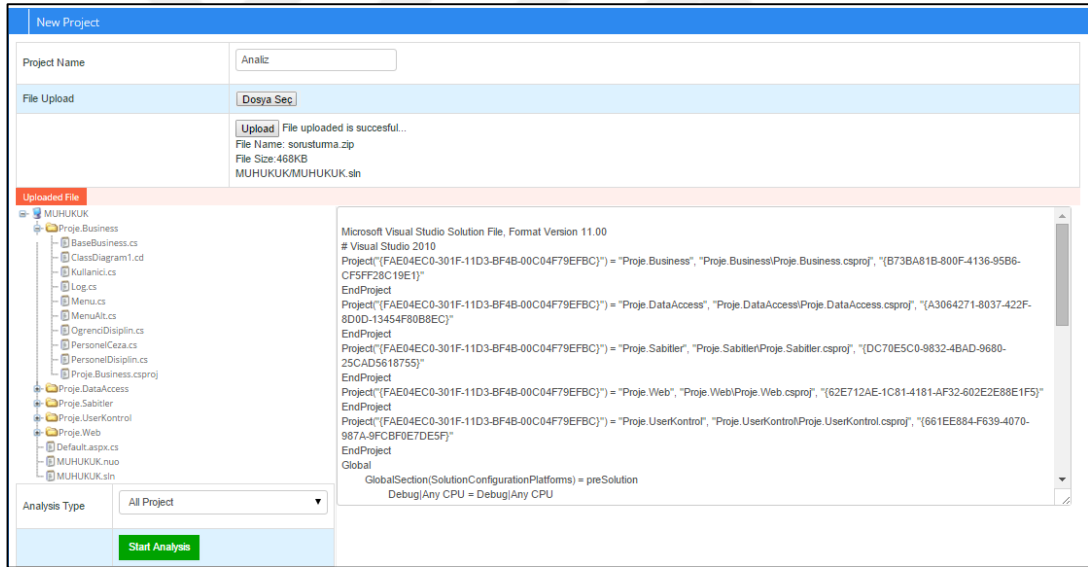
Project Name	File Name	LOC	SLOC	BLOC	Level	Score
Proje.Business	BaseBusiness.cs	0	0	0		
Proje.Business	Ogrenci.cs	0	0	0		
Proje.Business	AssemblyInfo.cs	0	0	0		
Proje.Business	BaseBusiness.cs	12	7	1		

Şekil 4.7. Online güvenlik seviyesi belirleme yazılımı-1



Şekil 4.8. Online güvenlik seviyesi belirleme yazılımı-2

Geliştirilen yazılım, kod metriklerini hesaplayarak kullanıcılara proje dosyalarındaki metriklerle ilgili bilgiler vermektedir. Sistemin çalışma yapısı dosya yükleme ve yüklenen dosyaların okunması esasına dayanmaktadır. Yüklenen dosyalar temizleme işleminden geçirilerek analize hazır hale getirilir. Temizleme işleminde resim, video, belge gibi dosyalar ayıklanmaktadır. Geliştirilen yazılım Asp.Net dilinde geliştirilmiş projelerin analizini gerçekleştirebilmektedir. Php ve Java gibi dillerdeki nesne yönelimli metriklerin hesaplanması sırasında yaşanan problemler ve bu dillerdeki farklı hesaplama yöntemleri sağlıklı veriler elde edilmesini engellemiştir. Bu sebeple sadece asp.net ile geliştirilen projeler güvenlik incelemesinden geçirilebilmektedir. Yüklenen dosyalar incelemiden geçtikten sonra tüm proje veya seçili dosyanın metrik hesaplaması arayüzden yapılabilir. Şekil 4.9’ da dosya yükleme ve metrik hesaplama arayüzü verilmiştir.



Şekil 4.9. Dosya yükleme ve metrik analizi

Online güvenlik yazılımında nesne yönelimli ve klasik metriklerden LOC, SLOC, BLOC, CLOC, CP, çevrimsel karmaşıklık (CC), bakım indexi (MIndex), halstead sayısı (HV), WMC ve NOM hesaplanmıştır.

Güvenlik derecelendirilmesi sırasında kullanılacak metrikler arasından, kod kalitesi ve kod karmaşıklığı açısından değerlendirilebilir olanlar seçilmiştir.

Çevrimsel karmaşıklık, bakım indexi ve sınıf başına düşen metot sayısı sınıflandırmada kullanılacak metriklerdir. Çevrimsel Karmaşıklık metriği için literatürde tespit edilen eşik değerleri çizelge 4.15’ de gösterilmiştir (Olague vd, 2007).

Çizelge 4.15. Çevrimsel karmaşıklık eşik değerleri

Kategori	Eşik Değer (x)	Risk İsimlendirmesi
1	$1 \leq x \leq 10$	Basit prosedürler, düşük risk
2	$11 \leq x \leq 50$	Orta Derecede Karmaşık, orta derece risk
3	$x \geq 50$	Test Edilebilir Değil, çok yüksek risk

Çizelge 4.16’ da sınıf başına düşen metotların eşik değerleri verilmiştir (Ramachandran, 2010).

Çizelge 4.16. NOM eşik değerleri

Kategori	Eşik Değer (x)	Risk İsimlendirmesi
1	$1 \leq x \leq 20$	Sınıf Karmaşıklığı için İyi bir Değer
2	$20 \leq x \leq 100$	Biraz Yüksek Karmaşıklık
3	$x \geq 100$	Yüksek Sınıf Karmaşıklığı

Çizelge 4.17’ de Bakım Endexine ait eşik değerlerin tablosu verilmiştir (Conor, 2007).

Çizelge 4.17. Bakım endexi eşik değerleri

Kategori	Eşik Değer (x)	Risk İsimlendirmesi
1	$0 \leq x \leq 9$	Düşük Bakım
2	$10 \leq x \leq 19$	Orta Seviyede Bakım
3	$20 \leq x \leq 100$	Yüksek Seviyede Bakım

Çizelge 4.15, 4.16 ve 4.17’ de verilen değerler metriklerin limitlerdeki riskli durumlarını ifade etmektedir. Geliştirilen online güvenlik yazılımında risk değerleri göz önüne alınarak programlardaki güvenlik seviyeleri belirlenmiştir.

Geliştirilen yöntemde puan bazlı derecelendirme yapılmıştır. Çizelge 4.18’ de metrik değerlerine karşılık verilen puanlar gösterilmiştir.

Çizelge 4.18. Eşik değer aralıklarına göre metrik puanları

Kategori	Eşik Değer Aralığı	Puan	Sınıflandırma Adı
1	$0 \leq x < 20$	0	Risk
2	$20 \leq x < 70$	50	Yarı Güvenli
3	$70 \leq x \leq 100$	100	Güvenli

Ölçümü yapılan proje dosyalarının her metriğine eşik değer aralıklarına göre bakım endex puanı, çevrimsel karmaşıklık puanı ve ağırlıklı metot sayısı puanı çizelge 4.18’ de göre ayrı ayrı verilmektedir. Projedeki metrik puanlarının ortalaması alınarak o proje dosyası ile ilgili genel bir puan hesaplanır. Eşitlik 4.4’ deki formül kullanılarak dosyaya ait puan hesaplanır.

$$SVI = \frac{\sum_{i=1}^n metrik_i}{n} \quad (4.4)$$

Eşitlik 4.4.’ deki n değeri, metrik türlerinin sayısını, metrik eşik değere göre hesaplanan metrik puanını, SVI ise Güvenlik değer endexini vermektedir. Çalışma kapsamında kod satırı barındıran tüm dosyalar inceleneceği için güvenlik düzeylerinin belirlenmesi için sınıflandırma yöntemi uygulanmıştır. Belirli bir özellik değerinin diğer özelliklere bağlı olmadan sonuca olan etkisini hesaplayan olasılıksal modelle dosyalar sınıflandırılmıştır. Öğreticili öğrenme yöntemiyle sisteme yüklenen dosyalar eğitim kümesine göre sınıflandırılır. Bu sınıflandırma yöntemi Naive Bayes Sınıflayıcısı olarak bilinmektedir. Algoritmanın temelinde öğretilmiş veriler üzerine dayandığı için başlangıçta bir eğitim seti belirlenmiştir. Eğitim seti herhangi bir test setinin belirli bir yüzdesi alınarak yapılabileceği gibi başlangıçta belirlenebilecek sabit değerler ile de belirlenebilmektedir. Öğretilmiş veri sayısının çok olması, test verilerinin gerçek kategorilerini tespit etme oranını arttırmaktadır. Çalışma kapsamında analizi gerçekleştiren proje dosyalarının ortalaması alınarak dosya sayısının %30 kadarı eğitim setinin sayısını oluşturmuştur. Sınıflandırma işlemi için bölüm 3.4.’ deki hesaplama yöntemi kullanılarak metrikler sınıflandırılmıştır. Oluşturulan eğitim setini kullanıcıların

New Analysis	* Security Level Training Set				
Security Level Training Set					
Bayes Classification					
	Cyclomatic Complexity	Maintainability Index	WMC	Score	Category
	0	0	0	0	Risk
	1	54	1	100	Secure
	12	21	19	83,33333333	Secure
	0	17	0	16,66666667	Risk
	18	28	22	66,66666667	Semi Secure
	52	115	0	0	Risk
	30	9	17	50	Semi Secure
	3	44,91032568	15	100	Secure
	53	19	5	50	Semi Secure
	10	26	64	83,33333333	Secure

Eğitim seti belirlendikten sonra Bayes sınıflandırma menüsü kullanılarak metrik analizi gerçekleştirilen projenin bayes sınıflandırması, ilgili analiz adı seçilerek gerçekleştirilir. Şekil 4.11’ de Bayes sınıflandırma arayüzü gösterilmektedir.

New Analysis

Security Level Training Set

Bayes Classification

* My Reports

Report Name

Analiz Proje

Classify

Run Bayes Classification

* Classification Results

File	Cyclomatic Complexity	Maintainability Index	WMC	Type	Result
Kullanici.cs	6,88090921481933E-18	2,77813224090003E-18	0,00130982408243099	Risk	3,38393868506438E-33
Kullanici.cs	0,0096692868364961	0,0094415651539402	0,0125413941898946	Semi Secure	0,309474609670008
Kullanici.cs	0,0159677353178998	0,0134142777914622	0,0185529848554036	Secure	0,690525390329992
Log.cs	6,88090921481933E-18	9,88928487097331E-19	0,00130982408243099	Risk	1,21435062514267E-33
Log.cs	0,0096692868364961	0,0093590025415829	0,0125413941898946	Semi Secure	0,309257497512723
Log.cs	0,0159677353178998	0,0133104942056804	0,0185529848554036	Secure	0,690742502487277
Menu.cs	6,88090921481933E-18	1,03134857306913E-17	0,00130982408243099	Risk	1,24410688921804E-32
Menu.cs	0,0096692868364961	0,00954467571721117	0,0125413941898946	Semi Secure	0,309830993318006
Menu.cs	0,0159677353178998	0,0135381849875173	0,0185529848554036	Secure	0,690169006681994
MenuAlt.cs	6,88090921481933E-18	3,77605629856759E-17	0,00130982408243099	Risk	4,51434497802119E-32

109

Bayes sınıflandırmanın temelinde olasılıksal hesaplamalar olduğu için sınıflandırma işleminde her bir metriğe göre analizi yapılan dosyaların güvenlik derecesi listede ayrı ayrı listelenir. Eşitlik 3.2.' deki formül uygulanarak en büyük değere ait güvenlik derecelendirmesinin risk kategorisi sonuç sınıflandırmayı vermektedir. Kullanıcının proje dosyalarına ait güvenlik seviyesi “Reports” menüsü tıklanarak görülebilmektedir. Reports sayfasında kullanıcının daha önce yapmış olduğu analizler ve güvenlik seviyeleri görülebilmektedir. Ayrıca satır seçilerek detaylı metrik raporu ve demografik gösterim incelenebilmektedir. Şekil 4.12’ de uygulamaya ait rapor sayfası gösterilmektedir.

*

My Reports

Report Name

Analiz Proje

*

Click Row for Metrics Detail

Project Name	File Name	LOC	SLOC	BLOC	Level	Score
Proje.UserKontrol	Label.cs	16	8	2	0,68634290739015	Secure
Proje.UserKontrol	LabelKulturlu.cs	84	48	8	0,595588624134792	Secure
Proje.UserKontrol	TextBox.cs	29	16	4	0,690980597454232	Secure
Proje.UserKontrol	TextBoxMetin.cs	29	12	3	0,690572642062069	Secure
Proje.Web	AssemblyInfo.cs	35	15	4	*	0
Proje.Web	Settings.Designer.cs	46	22	1	0,688651090860514	Secure
Proje.Web	UserKullaniciLogin.ascx.cs	75	39	4	0,653458182443847	Secure
Proje.Web	UserKullaniciLogin.ascx.designer.cs	114	13	1	*	0
Proje.Web	UserMenu.ascx.cs	57	38	1	0,62399732006684	Secure
Proje.Web	UserMenu.ascx.designer.cs	24	3	1	*	0

Şekil 4.12. Bayes sınıflandırma raporları

Güvenlik belirleme yazılımında kullanıcılar kod kalitesini oluşturan metriklerle eşik değerleri kullanarak dosyaların güvenlik seviyelerini belirleyebilmektedir.

4.3. Bulguların Değerlendirilmesi

Geliştirilen statik kod analiz yazılımı, web uygulamalarına ait girdi denetimi, metrik analizi ve stil denetimi yapmaktadır. Asp.Net, php ve java dillerinde geliştirilen uygulamalar analiz aracının dosya yükleme bölümü kullanılarak sisteme yüklenir. Uygulama arayüzü kullanılarak analiz için gerekli ayarlamalar

yapıldıktan sonra yazılım test ve değerlendirme aşaması gerçekleştirilir. Çalışma kapsamında geliştirilen test ve değerlendirme yazılımı kullanılarak 3 programlama dilinde 5’ er açık kaynak kodlu proje analiz edilmiştir. Test edilen projeler piyasada sık kullanılan araba satış sitesi, sosyal paylaşım sitesi, araba kiralama sitesi ve sanal pos uygulamaları gibi github’ dan indirilmiş açık kaynak projelerdir.

4.3.1. Girdi Denetimi Sonuçları

Uygulamalar test edilirken elde edilen sonuçlar riskli ve yarı riskli kategorilerine göre sınıflandırılmıştır. Örneğin, SQLI saldırılarında kod satırında sorgu varsa ve doğrudan parametre almış ise “riskli” kategorisinde değerlendirilmektedir. Sorgu olmayan satırlarda request nesnesi veya normal parametre kullanılmış ise “yarı riskli” kategorisinde değerlendirilmiştir. XSS zafiyetlerinde ise dışarıdan alınan parametre doğrudan ekrana bastırılıyorsa “riskli” kategorisinde, ekrana bastırılan yerde sadece string sabit değerkullanılıyorsa da “yarı riskli” kategorisinde değerlendirilmiştir. Diğer zafiyet tiplerinde de zafiyet oluşturan kod satırının kullanıldığı durumlara göre kategorizasyon yapılmıştır. Asp. Net ile geliştirilmiş ve piyasada kullanılan açık kaynak projelerde yapılan girdi analizi test çalışmasından elde edilen sonuçlar çizelge 4.19’ da verilmiştir.

Çizelge 4.19. Asp.net projelerinin girdi analizi sonucunda elde edilen test sonuçları

No	Proje Dili	Proje Adı	Zafiyet Türü	Yarı Riskli Satır Sayısı	Riskli Satır Sayısı	Toplam
1	Asp. Net	Araba Satış Sitesi	SQLI	0	23	23
			XSS	4	37	41
			Web Config	1	0	1
2	Asp. Net	Yorum Sitesi	SQLI	16	0	16
			XSS	1	1	2
			Ayar Dosyası	2	0	2

Çizelge 4.19. Asp.net projelerinin girdi analizi sonucunda elde edilen test sonuçları (Devam)

3	Asp. Net	Dershane Otomasyonu	SQLI	0	8	8
			XSS	1	1	2
			Yönlendirme	1	2	3
			Ayar Dosyası	1	0	1
4	Asp. Net	Online Çiçek Sitesi	SQLI	0	8	8
			XSS	1	1	2
			Yönlendirme	1	2	3
			Ayar Dosyası	1	0	1
5	Asp. Net	Online Nakliye Sitesi	SQLI	20	18	38
			XSS	89	491	578
			Serileştirme	0	18	18
			Yönlendirme	0	14	14
			Ayar Dosyası	0	16	16

Php ile geliştirilen projeleri test etmek amacıyla yurtdışında da kullanılan açık kaynak web projeleri kullanılmıştır. 5 adet projenin girdi analizi sonucunda elde edilen test verileri çizelge 4.20' de verilmiştir.

Çizelge 4.20. Php projelerinin girdi analizi sonucunda elde edilen test sonuçları

No	Proje Dili	Proje Adı	Zafiyet Türü	Yarı Riskli Satır Sayısı	Riskli Satır Sayısı	Toplam
1	Php	Sosyal Ağ Sitesi	SQLI	36	73	109
			XSS	55	216	271
			Check Executable	0	2	2
			File Validation	0	16	16
			File Inclusion	44	6	50
			Check evaluation	0	32	32
			Header Injection	0	33	33
			SQLI	39	14	16
			XSS	0	23	2
			File Validation	0	4	2
2	Php	Araba Kiralama Sitesi	Header Injection	0	2	

Çizelge 4.20. Php projelerinin girdi analizi sonucunda elde edilen test sonuçları(Devam)

3	Php	Sanal Pos Uygulaması	SQLI	148	4	152
			XSS	12	264	276
			File	3	122	125
			Inclusion			
			Check	18	0	18
			Evaluation			
			Header Injection	75	20	95
4	Php	Otel Rezervasyon Sis.	SQLI	39	16	55
			XSS	1	53	54
			Check	0	8	8
			Executable			
			File	2	2	4
			Inclusion			
			Check	0	2	2
5	Php	Instagram Benzeri Sis.	BackTick			
			SQLI	9	0	9
			XSS	0	1	578
			File	0	3	3
			Validation			
			File	0	2	2
			Inclusion			
			Header Injection	1	2	3

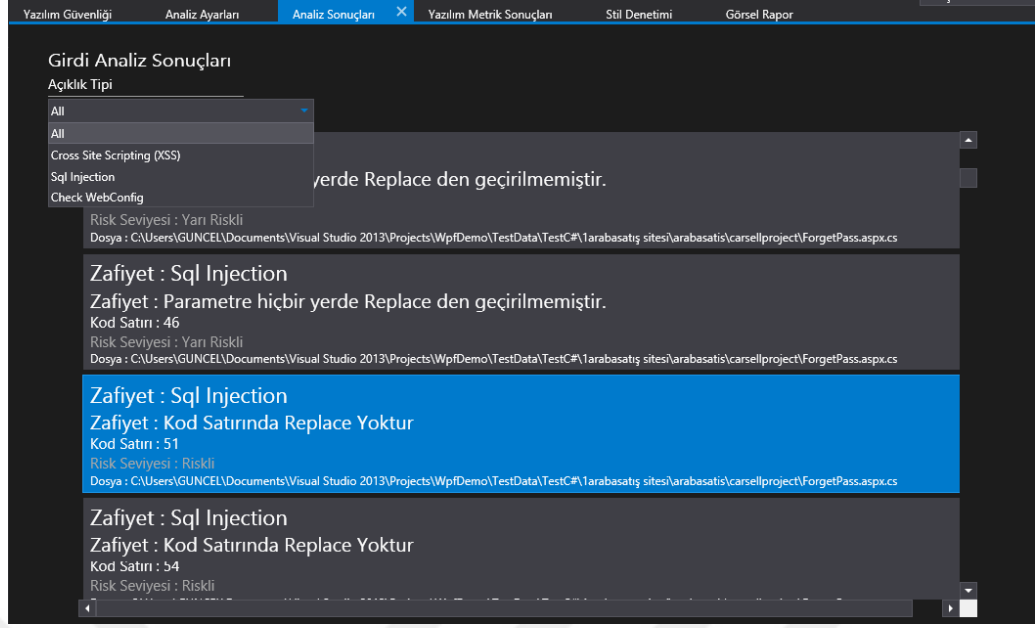
Php ile geliştirilen yazılımlarda zafiyetlerin yanında güvenlik kriterlerine uyan satırlarda tespit edilmiştir. 4 nolu otel rezervasyon sisteminde “mysql_real_escape_string” metodu 4 farklı satırda kullanılarak muhtemel güvenlik açıkları kapatılmıştır.

Java ile geliştirilen yazılımları test edebilmek adına github’dan indirilen açık kaynak web uygulamaları kullanılmıştır. Javada sık karşılaşılan güvenlik zafiyetlerinin test edildiği analiz aracında 5 farklı uygulama girdi denetiminden geçirilmiştir. Test sonucunda elde edilen sonuçlar çizelge 4.21’ de verilmiştir.

Çizelge 4.21. Java projelerinin girdi analizi sonucunda elde edilen test sonuçları

No	Proje Dili	Proje Adı	Zafiyet Türü	Normal Satır Sayısı	Yarı Riskli Satır Sayısı	Riskli Satır Sayısı	Toplam
1	Java	Yemek Sipariş Sitesi	SQLI	0	9	2	11
			XSS	0	0	27	27
			Servlet Check	2	0	0	2
			Modifiers	190	0	0	190
			Temp File	1	0	0	1
2	Java	Jsf Örnek Site	Servlet Check	2	0	0	2
			SQLI	0	1	7	8
			XSS	0	10	0	2
			Clone	0	0	1	1
			Modifiers	2	0	0	2
			Temp Files	0	0	1	1
			Privileged Block	3	0	4	7
			RequestDispatcher	0	0	4	4
			XEE Expansion	1	0	1	2
			Ldap Injection	0	0	2	2
			Xpath Injection	0	1	1	2
3	Java	Film Sitesi	Servlet Check	2	0	0	2
			SQLI	0	8	2	11
			XSS	0	13	23	36
			RequestDispatcher	0	0	2	2
4	Java	Öğrenci Bilgi Sis.	Servlet Check	3	0	0	3
			SQLI	0	0	38	54
			XSS	0	75	126	8
5	Java	Jsp Test	Servlet Check	14	0	0	14
			SQLI	0	18	1	19
			XSS	0	51	84	135
			Modifiers	2	0	0	2
			File Inclusion	0	0	2	2
			Header Injection	1	1	2	3

Geliştirilen analiz yazılımında girdi analiziyle satırlarda bulunan zafiyetler analiz sonuçları sekmesinde detaylarıyla birlikte görülebilmektedir. Şekil 4.13' de analiz sonuçları sayfası gösterilmiştir.



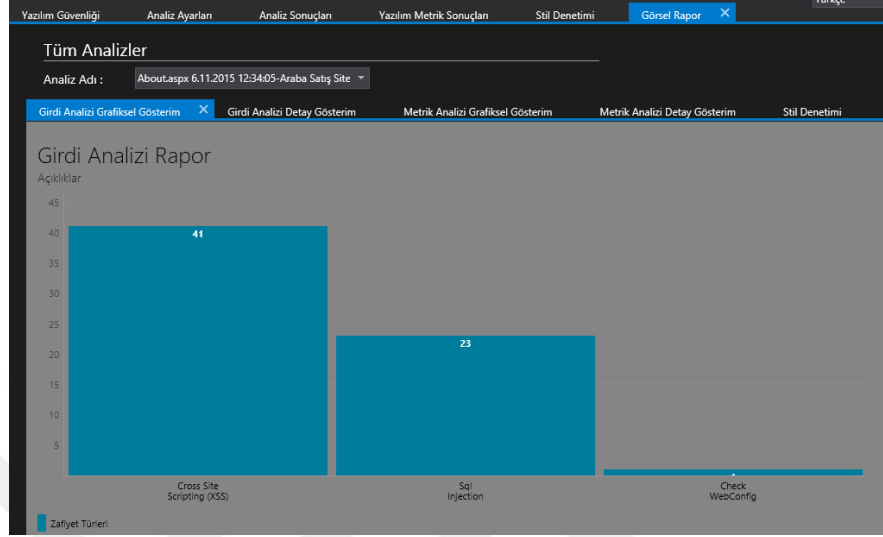
Şekil 4.13. Girdi analizi sonuç ekranı

Analiz sonucunda listelenen satırlara tıklanarak tespit edilen zafiyet ile ilgili detay penceresi açılmaktadır. Detay penceresinde zafiyet adı, zafiyet açıklaması, detay bilgiler, kod satırı, dosya adı ve risk seviyesi verilmektedir. Detay bilgilere ait pencere şekil 4.14’ de verilmiştir.



Şekil 4.14. Girdi analizi detay penceresi

Kullanıcıların önceki çalışmalarındaki girdi denetim detaylarını inceleyebilmeleri için görsel raporlar sayfası kullanıcılara grafiksel bir gösterim sunmaktadır. Şekil 4.15’ de girdi analizi rapor sayfası verilmiştir.



Şekil 4.15. Girdi analizi rapor sayfası

4.3.2. Metrik Analiz Sonuçları

Test ve Değerlendirme Yazılımının metrik analizi bölümünde 3 dilde 5 er tane web uygulamasının kaynak kod kalite analizi ölçülmüştür. Ölçümler sırasında nesne yönelimli, McCabe ve klasik metrik türleri ölçülmüştür. Çizelge 4.22’ de 3 dilde ölçülen metriklerin değerleri verilmiştir.

Çizelge 4.22. asp.net, php ve java projelerinin metrik analizi sonucunda elde edilen test sonuçları

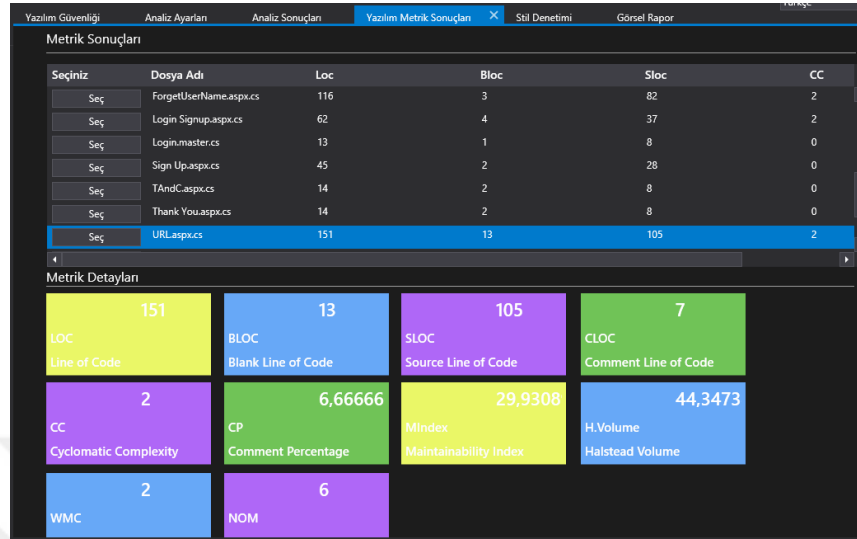
	Dil	Proje	LOC	SLOC	BLOC	CLOC	CC	CP	MI	HV	WMC	NOM
1	Asp. Net	Araba Satış Sitesi	871	566	53	77	15	%16	44,7	26,8	15	44
2	Asp. Net	Yorum Sitesi	105	64	9	19	2	37,6	55,3	30	2	4

Çizelge 4.22. asp.net, php ve java projelerinin metrik analizi sonucunda elde edilen test sonuçları (Devam)

3	Asp. Net	Dersha ne Sis.	239	134	17	42	5	40,8	49,52	33,5	5	8
4	Asp. Net	Çiçek Sitesi	4515	1324	330	418	80	78,88	52,09	30,7	80	101
5	Asp. Net	Nakliye Sitesi	29968	14948	2942	505	995	4,45	45,77	33,03	939,4	2900
6	Php	Sosyal Ağ Sitesi	2216	2033	4	179	59	17,71	-	-	-	-
7	Php	Araba Kiralama Sitesi	2273	2269	4	0	7	0	-	-	-	-
8	Php	Sanal Pos	8462	8347	10	105	99	5,59	-	-	-	-
9	Php	Otel Rezervasyon Sis.	2677	2481	143	53	27	5,11	-	-	-	-
10	Php	Instagram Benzeri Sis.	175	171	2	2	2	11,11	-	-	-	-
11	Java	Yemek Sipariş Sitesi	13905	12776	922	207	2245	1,1	-	-	5,85	596
12	Java	Jsf Örnek Site	776	660	78	38	86	6,21	-	-	4,1	51
13	Java	Film Sitesi	856	718	134	4	77	0,39	-	-	1,9	45
14	Java	Öğrenci Bilgi Sis.	2594	2386	171	37	456	1,82	-	-	0,5	9
15	Java	Jsp Çalışmaları	4479	3857	256	366	761	6,5	-	-	2,65	59

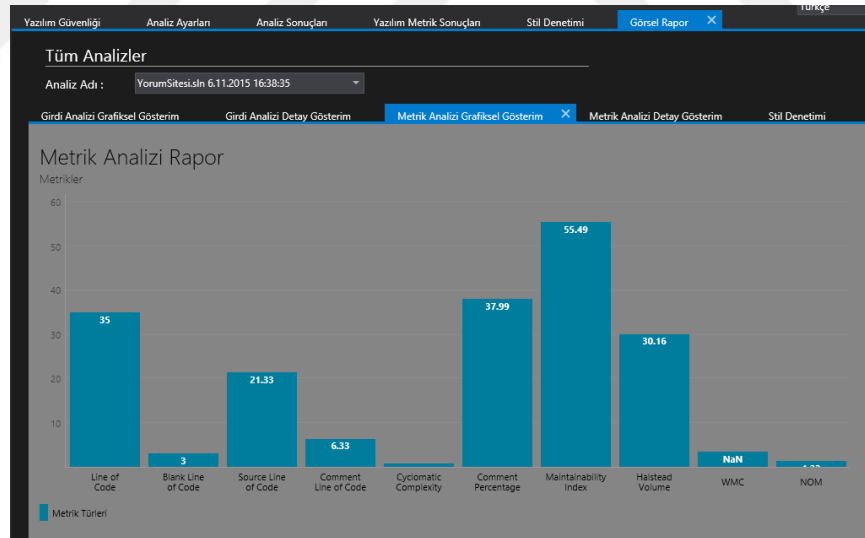
Metriklerin verildiği çizelge 4.22’ de dosya veya sınıf bazında ölçülen metriklerin proje genelindeki toplam değerleri verilmiştir. CP, WMC ve Mindex değerleri hesaplanırken projedeki ortalamaları alınarak hesaplanan değer tabloya eklenmiştir. Geliştirilen analiz yazılımında metrik analiziyle ölçülen kodlar analiz

sonuçları sekmesinde detaylarıyla birlikte görülebilmektedir. Şekil 4.16' da metrik analiz sonuçları sayfası gösterilmiştir.



Şekil 4.16. Metrik analizi sonuç ekranı

Metriklerin önceki çalışmalara ait rapor sayfası ise şekil 4.17' de gösterilmiştir.



Şekil 4.17. Metrik analizi rapor sonuç ekranı

4.3.3. Stil denetim sonuçları

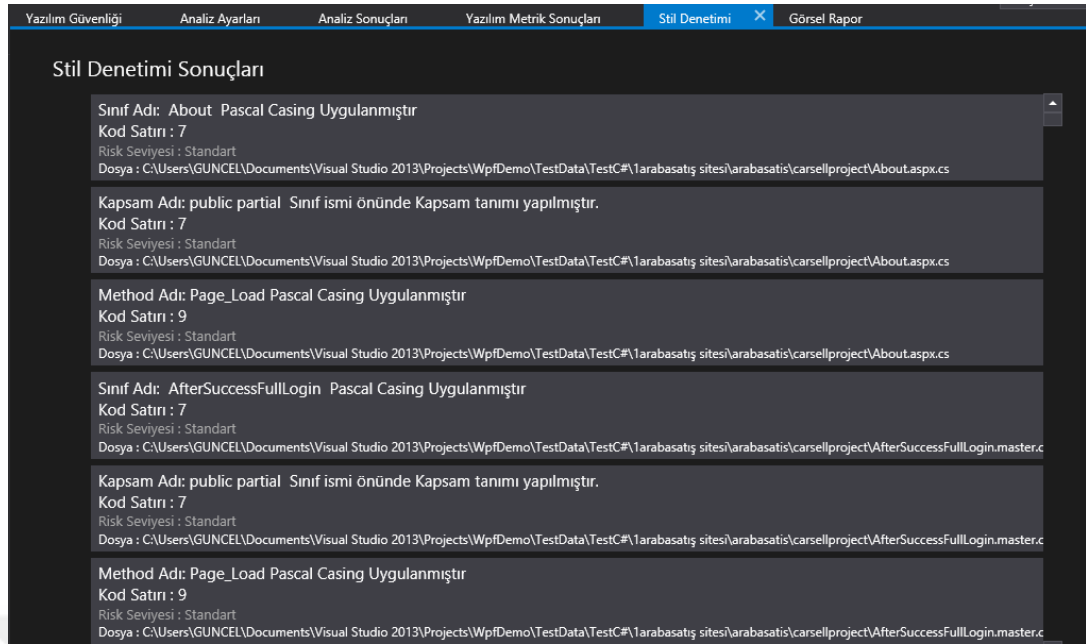
Stil denetimi projelerin statik kod analizi sırasında uygulanması gereken bir aşamadır. Özellikle kodlama standartlarının yazılımın her aşamasında

uygulanması projelerin gelişimi ve sürdürülebilirliği açısından statik analizin önemli bir bölümüdür. Çalışma kapsamında analiz aracının son bölümü stil denetimine ayrılmıştır. 3 dilde 5’ er web uygulaması geliştirilen analiz araçındaki stil denetiminden geçirilmiştir. Elde edilen test sonuçları çizelge 4.23’ de verilmiştir.

Çizelge 4.23. asp.net, php ve java projelerinin stil denetimi sonucunda elde edilen test sonuçları

No	Dil	Proje Adı	Normal Satır Sayısı	Yarı Riskli Satır Sayısı	Riskli Satır Sayısı
1	Asp.Net	Araba Satış Sitesi	197	0	40
2	Asp.Net	Yorum Sitesi	24	0	9
3	Asp.Net	Dershane Otomasyonu	59	0	8
4	Asp.Net	Online Çiçek Sitesi	459	0	255
5	Asp.Net	Online Nakliye Sitesi	2380	0	2349
6	Php	Sosyal Ağ Sitesi	4404	101	1283
7	Php	Araba Kiralama Sitesi	2663	129	620
8	Php	Sanal Pos Uygulaması	11804	256	2484
9	Php	Otel Rezervasyon Sis.	4573	120	968
10	Php	Instagram Benzeri Sis.	250	7	52
11	Java	Yemek Sipariş Sitesi	335	0	493
12	Java	Jsf Örnek Site	35	0	8
13	Java	Film Sitesi	61	0	19
14	Java	Öğrenci Bilgi Sis.	34	0	29
15	Java	Jsp Çalışmaları	178	0	213

Geliştirilen analiz yazılımında stil denetiminde ölçülen kodlar stil denetim sekmesinde detaylarıyla birlikte görülebilmektedir. Şekil 4.18’ de stil denetim sonuçları sayfası gösterilmiştir.



Şekil 4.18. Stil denetimi sonuç ekranı

Stil denetimi sonucunda standartlara uymayan satırlar ile ilgili dosya adı, kod satırı, risk seviyesi ve standart adı listede verilmiştir.

4.3.4. Online güvenlik analiz yazılım sonuçları

Yazılım kod metrikleri ile uygulamaların güvenlik derecelendirilmesi 5 asp.net uygulamasıyla gerçekleştirilmiştir. Metrik ölçüm sonuçlarına göre yapılan sınıflandırma ile projelerin güvenli, yarı güvenli ve riskli dosyaları tespit edilmiştir. Çizelge 4.24' de sınıflandırılan dosyaların sayıları verilmiştir.

Çizelge 4.24. Online güvenlik yazılımı test sonuçları

No	Dil	Proje Adı	Güvenli Dosya Sayısı	Yarı Güvenli Dosya Sayısı	Riskli Dosya Sayısı	Sınıflandırma Başarım oranı
1	Asp.Net	Test Projesi-1	18	7	3	%92,85
2	Asp.Net	Test Projesi-2	65	15	5	%87
3	Asp.Net	Dershane Otomasyonu	4	0	0	%100
4	Asp.Net	Online Nakliye Sitesi	53	32	0	%92,94
5	Asp.Net	Test Projesi-3	56	14	3	%86,3

Gerçekleştirilen testler sırasında kullanılan bazı test projelerinin sınıflandırma açısından istenen özelliklerde olmadığı ve başarılı bir sınıflandırma yapamadığı gözlenmiştir. Sınıflandırmada çevrimsel karmaşıklık, bakım indexi ve ağırlıklı metot değerleri kullanıldığı için bu değerlerin kullanılabilir olması gerekmektedir. Bu kapsamda daha önce asp.net testlerinde kullanılan projeler yerine 3 adet yeni test projesi eklenmiştir. Güvenlik değerlendirme yazılımında kullanıcılar önceki testlerine ait sınıflandırma sonuçlarını da web ortamında inceleyebilmektedirler.



5. SONUÇ VE ÖNERİLER

Teknolojik cihazların günlük hayatımızın vazgeçilmez bir parçası olması, beraberinde cihazlardaki yazılımsal veya donanımsal birimlerin kullanımını da yaygınlaştırmıştır. Bankacılık işlemleri, online alışveriş, anket çalışmaları, uzaktan eğitim, ödeme işlemleri gibi günlük hayatımızdaki birçok işlem internet siteleri veya mobil uygulamalar üzerinden yapılmaktadır. Gizliliğin önemli olduğu günümüzde fiziksel güvenlik yerini sanal güvenliğe bırakmıştır. Bu tür uygulamalardaki güvenlik zafiyetleri kullanıcıların maddi ve manevi açıdan zarara uğramalarına neden olabilmektedir. Donanımsal zafiyetlerin teknolojik cihazlarla önlemlerinin alınması, saldırganları yazılımlara yönlendirmiştir. Geliştiricilerin hızla artması ve yaşlarının düşmesi, serbest çalışarak para kazanma isteği, mobil uygulamaların yaygınlaşmasına ve zafiyet barındıran kodların çoğalmasına neden olmaktadır. Zafiyetlerin birçoğu ise geliştiricilerin bilinçsiz kod geliştirmesinden kaynaklanmaktadır.

Geliştirilen yazılımların güvenli olup olmadığı çeşitli yöntemlerle tespit edilebilmektedir. Dinamik ve statik olarak genelleştirilen testler ile uygulamalar kullanıma sunulmadan ve kullanım sırasında test edilebilmektedir. Dinamik testler ile kullanıma sunulan yazılımlar sızma testinden geçirilirler. Sızma testinde iyi niyetli sızma test uzmanı sistemi dışarıdan manipüle etmeye çalışır. Statik testlerde ise ürün henüz kullanıma sunulmadan geliştirme aşamasında test edilir.

Bu tez çalışmasında, geliştirilen yazılımların güvenli olup olmadığı test edilmiştir. Güvenlik analizinin yapılabilmesi için Yazılım Test ve Değerlendirme Aracı ile Online Güvenlik Analiz yazılımı geliştirilmiştir. Web ortamı için geliştirilen yazılımların incelenebildiği çalışmalarda güvenlik tespiti statik kod analiz yöntemi ile gerçekleştirilmiştir. Çalışmanın ilk bölümünde güvenlik hakkında bilgiler verilmiş ve web uygulamalarındaki zafiyetlerden bahsedilmiştir. İkinci bölümde ise bu alanda daha önce çalışılan konular ve geliştirilen uygulamalara değinilmiştir. Tezin materyal ve yöntem bölümünde statik kaynak kod analizinden bahsedilirken kategorileri hakkında bilgiler verilmiştir. Web

uygulama güvenliği konusunda bilgiler verildikten sonra, karşılaşılan zafiyetlerden bahsedilmiştir. Kaynak kod metrikleri ve Bayes sınıflandırma algoritmaları hakkında bilgiler verilmiştir. Araştırma bulguları bölümünde test projelerinin analizini yapmak için test ve değerlendirme yazılımı, güvenlik derecelendirmesi yapmak için de online güvenlik yazılımı geliştirilmiştir. Geliştirilen bu yazılımlar örnek projelerle test edilmiş ve elde edilen sonuçlar tablolar halinde verilmiştir. Çalışmanın sonuç bölümünde ise elde edilen bulgular hakkında değerlendirmeler verilerek yazılım güvenliği için yapılması gerekenler anlatılmıştır.

Çalışma kapsamında test ve değerlendirme yazılımı masaüstü, güvenlik derecelendirmesi ise web uygulaması olarak geliştirilmiştir. Test ve değerlendirme yazılımı Visual Studio platformunda Wpf teknolojisiyle geliştirilmiştir. Veritabanı olarak MsSql Server programı kullanılmıştır. Yazılım, kullanıcıların detaylı analizler gerçekleştirebilmesi ve raporlama yapabilmesi amacıyla 5 sekmeye ayrılmıştır. Genel analiz ayarlarının yapıldığı bölümün yanında analiz detayları farklı sekmelerde girdi analizi, metrik analizi ve stil denetimi olarak gösterilmektedir. Son sekmede ise geçmişte yapılan analizlerin detaylı ve grafiksel olarak raporlaması verilmiştir. Online güvenlik derecelendirme yazılımı ise Visual Studio platformunda Asp.Net teknolojisi kullanılarak geliştirilmiştir. Veritabanı olarak MsSql Server programı kullanılmıştır. Yazılım web tabanlı bir uygulama olmasından dolayı kullanıcı doğrulamasıyla kullanılabilir. Analizin tamamlanabilmesi 2 aşamada gerçekleşmektedir. İlk aşamada projelerin genel metrik hesaplamaları yapılmaktadır. İkinci aşamada hesaplanan metrikler Naive Bayes Sınıflandırma algoritmasıyla risk gruplarına göre kategorize edilmektedir. Geçmişe dönük hesaplamalarda proje kapsamında arayüzden takip edilebilmektedir.

Geliştirilen test ve değerlendirme yazılımı asp.net, php ve java dillerinde geliştirilen projeleri güvenlik testinden geçirmektedir. Güvenlik testi yapılırken statik kod analiz yöntemi kullanılmıştır. Kod analizi kapsamında girdi denetimi, metrik analizi ve stil denetimi uygulanarak projelerin analizi gerçekleştirilmektedir. Çalışma ile asp.net, java ve php dillerinde, internetten

indirilen 5' er açık kaynak web projesi girdi denetimi, metrik analizi ve stil denetimi aşamalarında test edilmiştir.

Girdi denetimi sonucunda uygulamalar geliştirildikleri dilde sık rastlanan zafiyetlere göre analiz edilmiştir. 3 dilde de SQLI, XSS saldırısı, ayar dosyası, dosya manipölasyonu, header enksiyonu sık rastlanan güvenlik zafiyetleridir. Çizelge 5.1, 5.2 ve 5.3' de test edilen projelerin toplam zafiyet sayılarının dillere göre dağılımı verilmiştir.

Çizelge 5.1. Asp.net test projelerinde zafiyet barındıran toplam satır sayısı

Proje Dili	Zafiyet Türü	Yarı Riskli Satır Sayısı	Riskli Satır Sayısı	Toplam
Asp.Net	SQLI	36	57	93
	XSS	96	531	627
	Ayar Dosyası	5	16	21
	Yönlendirme	2	18	20
	Serileştirme	0	18	18

Çizelge 5.2. Php test projelerinde zafiyet barındıran toplam satır sayısı

Proje Dili	Zafiyet Türü	Yarı Riskli Satır Sayısı	Riskli Satır Sayısı	Toplam
Php	SQLI	271	107	378
	XSS	68	557	625
	Check Executable	0	10	10
	File Validation	0	23	23
	File Inclusion	49	132	181
	Check evaluation	18	32	50
	Header Injection	76	57	133
	Check BackTick	0	2	2

Çizelge 5.3. Java test projelerinde zafiyet barındıran toplam satır sayısı

Proje Dili	Zafiyet Türü	Yarı Riskli Satır Sayısı	Riskli Satır Sayısı	Toplam
Java	SQLI	36	50	86
	XSS	149	260	409
	Clone	0	1	1
	Temp Files	0	1	1
	Privileged Block	0	4	4
	RequestDispatcher	0	6	6
	XEE Expansion	0	1	1
	Ldap Injection	0	2	2
	Xpath Injection	1	1	2
	Header Injection	1	2	3

Çizelge 5.1, 5.2 ve 5.3’ de verilen test sonuçlarına göre sql enjeksiyonu ve XSS zafiyetlerinin dil farkı gözetmeksizin tüm uygulamalarda var olduğu tespit edilmiştir. Diğer zafiyetlerin ise uygulama diline ve projelerin büyüklüğüne göre değişmekte olduğu gözlemlenmiştir.

Girdi denetimlerinde elde edilen test sonuçları hakkında karşılaştırmalı bilgi sahibi olmak ve tez çalışması sırasında geliştirilen analiz aracının geçerliliğini test etmek adına asp.net, java ve php de geliştirilen 5’ er test projesi piyasada kullanılan yazılımlarla test edilmiştir. Asp.Net projeleri için piyasada kullanılan VisualCodeGrapper (SourceForge, 2015) ve AppCodeScan (Blueinfy: Tools, 2010), php projeleri için Rips (SourceForge, 2011) ve VisualCodeGrapper (SourceForge, 2015), Java projeleri için AppCodeScan (Blueinfy: Tools, 2010) yazılımları kullanılmıştır. Asp.Net test projeleri için AppCodeScan ve VisualCodeGrepper yazılımlarında elde edilen test sonuçları Çizelge 5.4’ de verilmiştir.

Çizelge 5.4. Asp.net test projelerinde zafiyet barındıran toplam satır sayısı

Proje Dili	Zafiyet Türü	AppCodeScan Yazılımı	VisualCodeGrepper Yazılımı
Asp.Net	SQLI	70	3
	XSS	170	10
	Ayar Dosyası	0	7
	Yönlendirme	44	23
	Girdi Denetimi	29	0

Java’ daki test projeleri için sadece AppCodeScan yazılımı kullanılmıştır. Elde edilen test sonuçları Çizelge 5.5’ de verilmiştir.

Çizelge 5.5. Java test projelerinde zafiyet barındıran toplam satır sayısı

Proje Dili	Zafiyet Türü	AppCodeScan Yazılımı
Java	SQLI	83
	XSS	281
	XPath Enjeksiyonu	2
	Yönlendirme	40
	File Inclusion	28

Php’ deki test projeleri için VisualCodeGrepper ve Rips yazılımları kullanılmıştır. Elde edilen test sonuçları Çizelge 5.6’ da verilmiştir.

Çizelge 5.6. Php test projelerinde zafiyet barındıran toplam satır sayıları

Proje Dili	Zafiyet Türü	Rips Yazılımı	VisualCodeGrepper Yazılımı
Php	SQLI	141	1
	XSS	690	68
	Code Execution	54	2
	Commad Execution	0	6
	File Inclusion	166	0

Çizelge 5.4, 5.5 ve 5.6’ daki sonuçlar incelendiğinde, son yıllarda ortaya çıkan zafiyetlerin tespit edilemediği, bulunan zafiyetlerin ise çalışma kapsamında geliştirilen yazılımın tespit ettiği zafiyet sayılarına göre daha az olduğu tespit edilmiştir.

XSS zafiyetlerinin 3 dilde de SQLI’ dan fazla olması saldırıların XSS üzerinde yoğunlaşma ihtimalinin daha fazla olduğunu ve XSS’ in web uygulamalarında dikkat edilmesi gereken bir açıklık olduğunu göstermektedir. Her yıl yayınlanan güvenlik raporundaki XSS zafiyet oranının yüksek olması, gerçekleştirilen testin doğruluğunu göstermektedir.

Metrik analizi sonucunda 3 dilde tespit edilen metrik sonuçlarına göre projeler kaynak kod kalitesi bakımından yorumlanabilmiştir. Metrik analizi sınıf veya dosya bazında yapılmaktadır. Analiz aracında da test projelerinin tüm dosyaları ayrı ayrı test edilmiştir. Analiz sonuçlarına göre elde edilen test sonuçları php dilinde geliştirilen uygulamalarının karmaşıklığının asp.net ve java dillerinde geliştirilen projelerin karmaşıklığına göre daha düşük olduğu gözlemlenmiştir. Asp.Net ve Java dillerindeki karmaşıklık değerlerinin yüksek çıkması bu dillerin nesne yönelimli mimariye daha yatkın olması ve method kullanımının daha çok olmasından da kaynaklanmaktadır. Ayrıca, karmaşıklık değerleri projenin büyüklüğü ve satır sayılarına göre değişmektedir.

Stil denetimi ile 3 dilin kendilerine ait kodlama standartları dikkate alınarak gerçekleştirilen analizde standartlara uymayan satırlar tespit edilerek kullanıcılara arayüzde rapor olarak verilmektedir. Stil denetimlerine dikkat edilmemesi, projenin sürdürülebilirliğini ve bakımını olumsuz yönde etkilemektedir. Analizler sonucu projelerdeki standart dışı kullanımlar analiz aracında geriye dönük olarak detaylı bir şekilde gösterilmektedir.

Online güvenlik analiz yazılımı kullanıcılara kaynak kod kalitesi hakkında bilgi vermek ve kullanıcılara yazılım güvenliği hakkında bilinçlendirmek amacıyla web tabanlı uygulama olarak geliştirilmiştir. Geliştirilen yazılımda asp.net dilinde geliştirilen yazılımlar derleme aşamasındayken sisteme yüklendikten sonra metrikler hesaplanır ve sistem tarafından güvenlik derecelendirmesi yapılır. Güvenlik derecelendirilmesi çevrimsel karmaşıklık, bakım indexi ve sınıf başına düşen metot sayılarının sınıflandırılmasıyla yapılmaktadır. Bu metriklerin değerleri sınıfın veya metotun kod bakımı veya kalitesi hakkında bilgiler vermektedir. Sınıflandırma sırasında metriklerin eşik değerlerine puan verilerek risk kategorizasyonu yapılmaktadır. Test projelerinin analizi sonucunda sınıflandırılan proje dosyalarının sınıflandırma başarıları ölçüldüğünde %100 ile %85 arasında bir oran tespit edilmiştir. Eğitim setindeki verilerin arttırılmasıyla sınıflandırma başarı oranının arttığı yapılan denemeler sonucunda ortaya çıkmıştır. Bu çalışmayla birlikte online güvenlik analiz yazılımında kaynak kod

metrikleri kullanılarak sınıf veya dosyalar üzerinde güvenlik derecelendirilmesinin yapılabileceği tespit edilmiştir.

Geliştirilen analiz araçlarıyla test projelerinde tespit edilen güvenlik zafiyetlerini engellemek için programlama dillerinin önlemleri almak adına geliştirdikleri hazır metotlar bulunmaktadır. Bunun yanısıra meydana gelen güvenlik açıklarının en büyük sebebi web uygulamalarındaki girdi problemidir. Bu girdi problemlerine geliştiricilerin hazırladığı metotlarda ek olarak engel olabilmektedir. Asp.net ve Java ile geliştirilen projelerde .Replace metodu içerisinde istenmeyen karakterler temizlenerek geriye problem yaratmayacak ifadeler oluşturulabilmektedir. Php de ise str_replace ve mysql_real_escape_string fonksiyonları html karakterleri temizlemektedir. Ek olarak geliştiriciler kendilerine ait temizleme (sanitize) fonksiyonlarını hazırlayarak girdi denetimsizliğini ortadan kaldırabilmektedirler. Beyaz liste denetimi yapılarak iyi karakterlerin veya karakter dizgilerinin kabul edilmesi veya düzenli ifadelerle girdi denetimi yapılabilmektedir. Kara liste denetimi ise girdi kontrolü için tavsiye edilmeyen bir uygulamadır. Sadece kötü karakterlerin kabul edilmemesi üzerine kurulan bir yapıdır. Fakat beraberinde listede olmayan ifadelerin filtreden geçmesi ve farklı atlatma yöntemlerinin uygulanması engellenememektedir. Geliştiriciler arasında yaygın bilinen bir yanıştır. Beyaz liste ve kara liste mantığının birarada kullanılması en doğru uygulama olmaktadır. Temizleme işlemi (sanitize) uygulanarak girdi kabul edilebilir bir formata çevrilmektedir. Beyaz ve kara liste mantığına göre çalışmaktadır. Örneğin telefon numarası içinde, sayı olmayan bütün karakterlerin silinmesi beyaz liste mantığı ile arıtmadır. Fakat tüm virgüllerin ayrılması kara liste mantığı ile arıtmaya örnektir. Şifreleme (Encoding) işlemi ile özel karakterler başka bir formata çevrilmektedir. Hedef yorumlayıcı için özel karakterlerin kodlama işlemi sonrası önemlerini yitirmiş olmaları amaçlanmaktadır. “Escape” işlemi çıktı denetimi kontrolü olarak da bilinmektedir. Yorumlayıcı girdiyi çalıştırmadan önce karakter değişikliği yapılarak olası sorgu değişikliğinin önüne geçilebilmektedir.

SQLI' nın önüne geçebilmek için geliştirilen yazılımdaki kontrol fonksiyonunda beyaz liste denetimi yapılmaktadır. Beyaz listenin yanında geliştirilen platforma bağlı olarak kullanılan hazır fonksiyonlarda kontrol edilmektedir. Java dilinde preparestatement fonksiyonu ve parametrelerin set edilmesi, php dilinde prepare("query") ve bind_param ile parametrelerin set edilmesi ve asp.net de parameters.addwithvalue fonksiyonunun kullanılması kontrol edilerek projelerdeki olası zafiyet satırları tespit edilmektedir.

XSS zafiyetlerini engelleyebilmek adına oluşturulan beyaz liste denetimi yanında asp.net ile geliştirilen projelerde RequestValidation özelliği kontrol edilir. <%@ Page validateRequest="false" %> ile zararlı kodların enjekte edilmesi engellenmiş olunur. Tüm projede kapatmak için web config dosyasında <pages validateRequest="false" /> ve <httpRuntime requestValidationMode="2.0"/> kontrol edilir. Asp.net 4.0 ve sonrası için bu parametrelerin kontrolü yeterli olmamaktadır. Literal control' de Mode ayarının "Encode" yapılmasıda XSS önlemini sağlamış olacaktır. Ayrıca Anti XSS kütüphanesi de alınabilecek önlemlerden biridir. Kod satırlarında <%=Microsoft.Security.Application.Encoder.HtmlEncode("");%> satırı kontrol edilmektedir. AntiXss kütüphanesinde htmsanitizationLibrary kullanılarak html kod içeren metinler zararlı kodlardan arındırılabilir. String metin=Microsoft.Security.Application.Sanitizier.GetSafeHtml(htmltext); ve GetSafeHtmlFragment("") fonksiyonları proje kapsamında XSS kontrol fonksiyonunda incelenmektedir. Java da geliştirilen projelerin kontrolü için beyaz liste denetiminin yanında Owasp-Esapi projesi kapsamında geliştirilen hazır kütüphanelerin fonksiyonları da kontrol edilmektedir. <%Esapi.validator().isValidInput();%> <%Esapi.encoder().encodeForHTML()%>, <%Esapi.encoder().encodeForJavaScript()%> ve <%Esapi.validator().getValidSafeHTML()%> kontrol edilen fonksiyonlardır. Php için de beyaz liste kontrolü en önemli kuraldır. Doğrulama ve arıtma için php de hazır olarak gelen filtre metodları kullanılmaktadır. Filter_input metodu yardımıyla get, post, cookie, env, sunucu kaynaklarından gelen veriler doğrulama işleminden geçirilip geçirilmediği kontrol edilir. Htmlspecialchars fonksiyonunun kontrolü ile de zararlı kodların eklenmesi engellenmiş olunur.

CSRF saldırılarının kontrolü için http referer başlığı kontrol edilmektedir. Java projeleri için Owasp.CSRFGuard kütüphanesi, Asp.Net için Viewstate özelliğinin aktive edilip edilmediği ve Web config üzerinde <pages enableviewstate="true" enableViewStateMac="true"/> ve AntiForgeryToken satırlarının olup olmadığı kontrol edilmektedir.

Komut enjeksiyonu saldırılarını önlemek için beyaz liste denetimi yapılmaktadır. Java için Owasp-Esapi projesi kontrol edilmektedir. Esapi kullanılmamışsa dilin match ve replace metodlarına bakılmaktadır. Php için escapeshellarg() ve escapeshellcmd metodlarına bakılarak, girdilerin escape işleminden geçirilip geçirilmediği kontrol edilmektedir.

Statik kod analiziyle uygulamalardaki güvenlik açıkları geliştirme aşamasında girdi denetimleriyle önlenebilmektedir. Geliştiriciler dışarıdan gelebilecek saldırılara karşı girdi alanlarını kontrol altında tuttuğu takdirde zafiyetlerin önüne büyük ölçüde geçebilmektedir. Ayrıca kod kalitesinin gelişimi de orta vadede güvenliği etkileyebilecek bir faktördür. Çevrimsel karmaşıklık değerinin 20 den büyük olması veya bakım endeksinin eşik değerlerin dışında olması kod gelişimini ve bakımını güçleştirmekte, karmaşık kodların olması hafıza taşması gibi güvenlik zafiyetlerine neden olabilmektedir. Kodlama standartlarındaki ihlal de kod bakımını ve sürdürülebilirliğini etkilemekte ve güvenliği dolaylı yoldan etkilemektedir.

Tez çalışmasında web uygulamalarında bulunan güvenlik zafiyetlerinin tespit edilmesi amaçlanmıştır. Bu kapsamda geliştirilen yazılımlar hem statik kod analiziyle girdi ve stil denetimi, hemde naive bayes ile proje dosyalarının güvenlik sınıflandırılmasını yapmışlardır. Çalışmanın siber güvenliğin son yıllarda artan önemiyle birlikte özellikle ulusal alanda yazılım güvenliğinin yeteri kadar dikkate alınmaması, bilinçli geliştiricilerin az olması ve güvenlik analiz yazılımlarının eksikliği göz önüne alındığında literatüre önemli bir katkı yapacağı düşünülmektedir. Geliştirilen analiz araçları hem test yazılımları olarak programcılar tarafından hemde eğitim aracı olarak güvenlik kurslarında

kullanılabilecek şekilde tasarlanmıştır. Bu çalışma kapsamında yazılım firmaları geliştirdikleri yazılımları test ederek güvenli yazılım geliştirme standartlarını yükseltebilir ve proje maliyetlerini azaltabilirler. İlerleyen süreçte, yazılım değerlendirme aracı için yapay zeka teknikleri ve yeni çıkan zafiyet türlerinin sisteme eklenerek daha hızlı ve doğru analizlerin yapılması planlanmaktadır. Ayrıca mobil yazılımlarında benzer yöntemlerle zafiyetlerin önlenileceği düşünülmektedir.



KAYNAKLAR

- Ademoğlu, C., Akyüz, D., 2008. Yazılım Geliştirmede Statik Kod Analizi. Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, 9-10 Ekim, İstanbul.
- Akoğlu, A., 2008. Php-\$_Get Methodu. Erişim Tarihi:13.12.2014. <http://www.serpito.com/php-get-metodu>.
- Akbulut, S., 2006. Veri Madenciliği Teknikleri ile Bir Kozmetik Markanın Ayrılan Müşteri Analizi ve Müşteri Segmentasyonu. Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 103s, Ankara.
- Akgün, F., Buluş, E., Buluş, H.N., 2005. Yazılım Mühendisliği Açısından Uygulamalardaki Ara Bellek Taşması Zafiyetinin İncelenmesi. Elektrik-Elektronik-Bilgisayar Mühendisliği 11. Ulusal Kongresi ve Fuarı, 22-25 Eylül, İstanbul.
- AL-Amro, H., El-Qawasmeh, E., 2012. Discovering Security Vulnerabilities And Leaks In ASP.NET Websites. Cyber Security, Cyber Warfare and Digital Forensic (CyberSec). 26-28 Haziran, Kuala Lumpur, 329-333.
- Alparslan, E. 2009. Güvenli Yazılım Geliştirme Modelleri. Erişim Tarihi: 10.08.2013. <http://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/guvenli-yazilim-gelistirme-modelleri.html>
- Alpaydın, E., 2000. Zeki Veri Madenciliği: Ham Veriden Alınan Bilgiye Ulaşma Yöntemleri. Bilişim 2000 Veri Madenciliği Eğitim Semineri, İstanbul.
- Altinkurt, D., 2012. Asp.Net'te Çerez (Cookie) Kullanımı. Erişim Tarihi: <http://daltinkurt.com/Icerik/216/ASP-NETte-Cerez--Cookie--Kullanimi.aspx>.
- Bilgi Güvenliği Akademisi, 2014. Web Uygulama Güvenlik Testleri Eğitimi, 450s.
- Blueinfy: Tools, 2010. Application Code Scanning and Tracing tool. Erişim Tarihi: 25.07.2015. <http://www.blueinfy.com/tools.html>.
- Buzluca, F., 2012. Yazılım Metrikleri. Erişim Tarihi:30.10.2014. akademi.itu.edu.tr/buzluca.
- Buzluca, F., 2013. Tasarım Kusurlarının Belirlenmesi. Erişim Tarihi:10.11.2014. akademi.itu.edu.tr/buzluca.
- Büchler, M., Oudinet, J., Pretschner, A., 2012. SPaCiTE-Web Application Testing Engine. IEEE Fifth International Conference on Software Testing, Verification and Validation. 17-21 Nisan, Montreal, 858-859.

- Calp, H., Arıcı, N., 2011. Nesne Yönelimli Tasarım Metrikleri ve Kalite Özellikleri İlişkisi. Politeknik Dergisi, 14(1), 9-14.
- Chess, B., West, J., 2007. Secure Programming with Static Analysis. Pearson Education, 559, Indiana, USA.
- Conor, M., 2007. Maintainability Index Range and Meaning. Erişim Tarihi: 13.12.2014.
<http://blogs.msdn.com/b/codeanalysis/archive/2007/11/20/maintainability-index-range-and-meaning.aspx>.
- Costa, M., Castro, M., Zhou, L., Zhang, L., Peinado, M., 2007. Bouncer: securing software by blocking bad input. 21. ACM SIGOPS symposium. 14-17 Ekim, Stevenson, Washington.
- Çebi, Ç. 2013. Bilgi Güvenliği. Erişim Tarihi: 20.07.2013.
http://www.cagataycebi.com/security/bilgi_guvenligi.pdf.
- Çeri, Y., 2009. Web'deki BÜYÜK Tehlike: CSRF. Erişim Tarihi: 06.01.2015.
<https://www.bilgiguvenligi.gov.tr/web-guvenligi/webdeki-buyuk-tehlike-csrf.html>.
- Dahse, J., 2010. RIPS-A static source code analyser for vulnerabilities in PHP scripts. Horst Görtz Institute Ruhr-University, 19s.
- Dewhurst, R., 2012. Implementing Basic Static Code Analysis into Integrated Development Environments (IDEs) to Reduce Software Vulnerabilities. University of Northumbria, Proje Raporu, 86s, Newcastle.
- Daylak, B., 2014. Php Dersleri. Erişim Tarihi: 09.11.2014.
http://www.dijitalders.com/icerik/14/362/_globals.html#.VJgy58gOA.
- Değerli, O., 2012. Naive Bayes Yöntemi ile Blog İçeriklerinin Sınıflandırılması. Gazi Üniversitesi, Bilişim Enstitüsü, Yüksek Lisans Tezi, 75s, Ankara.
- Demir, B., 2013. Yazılım Güvenliği Saldırı ve Savunma, Dikey Eksen Yayın Dağıtım, 423s, İstanbul.
- Dirik, S., 2011. Yazılım Ölçümleme. Erişim Tarihi: 10.01.2015. <http://serhatdirik.blogspot.ru/2011/04/yazlm-olcumleme.html>.
- Eklöf, R., 2011. Improving software development with static code analysis in a traceable environment. KTH Industrial Engineering and Management, Yüksek Lisans Tezi, 59s, Stockholm.
- El-Emam, K., 2002. Object-oriented metrics: A review of theory and practice. Ślezak, D. Kim, T.-h. Kiumi, A. Jiang, T. Verner, J. Abrahao, S. (Ed), in Advances in Software engineering İçinde (23-50), Springer, 347, Verlag New York, Inc.

- Erdemir, U., Tekin, U., Buzluca, B., 2008. Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi. Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, İstanbul, Türkiye.
- Erdik, C.F., 2014. Owasp Top 10: CSRF - Güvensiz Doğrudan Nesne Erişimi Zafiyeti. Erişim Tarihi: 05.05.2014. <http://blog.bga.com.tr/2014/02/owasp-top-10-csrf-guvensiz-dogrudan.html>.
- Erdogan, G., 2009. Security Testing of Web Based Applications. Norwegian University of Science and Technology, Department of Computer and Information Science, Yüksek Lisans Tezi, 171s, Norveç.
- Jovanovic, N., Kruegel, C., Kirda, E., 2006. Pixy: a static analysis tool for detecting web application vulnerabilities., Security and Privacy, IEEE Symposium, 21-26 Mayıs, Berkeley/Oakland, CA, 263-269.
- Gallagher, T., Jeffriers, B., Landuer, L., 2006. Hunting Security Bugs. Microsoft Press, 586, Washington.
- Garg, A., 2014. An approach for improving the concept of Cyclomatic Complexity for Object-Oriented Programming. Erişim Tarihi:24.10.2014. <http://arxiv.org/abs/1412.6216>.
- Göksu, S., 2009. ASP.NET Caching(Önbellekleme) – Data Caching. Erişim Tarihi: 29.10.2014. <http://www.semgoksu.com/aspnet-cachingonbellekleme-%E2%80%93-data-caching-yazisi/508.aspx>.
- Gürsoy, İ., 2011. Application State Kullanımı. Erişim Tarihi: 04.11.2014. <http://www.ismailgursoy.com.tr/application-state-kullanimi>.
- Güvercin, E., 2013. Asp.Net ile Session. Erişim Tarihi: 28.11.2014. <http://www.erenguvercin.com/2013/02/aspnet-ile-session.html>.
- Huming, Y., Jones, N., Bullock, G., Yuan, Y. Y., 2011. Teaching Secure Software Engineering: Writing Secure Code. Software Engineering Conference in Russia (CEE-SECR), 7th Central and Eastern European. 31 Ekim-3 Kasım, Moskova, 1-5.
- Ilgin, I., 2008. HTTP-GET DDoS Atak Engelleme. Erişim Tarihi:25.11.2014. <https://www.bilgiguvenligi.gov.tr/web-guvenligi/http-get-ddos-atak-engelleme.html>.
- Kalay, A., 2009. Statik Kod Analizinin Yazılım Geliştirme Sürecindeki Yeri ve Yazılım Kalitesine Etkisi, 4. Ulusal Yazılım Mühendisliği Sempozyumu. 8-10 Ekim, İstanbul.

- Karadeniz, A., 2012. Asp.Net Veri Gönderme Yöntemleri. Erişim Tarihi: 25.09.2014. <https://alperkaradeniz.wordpress.com/2012/05/03/asp-net-veri-gonderme-yontemleri/>.
- Karagedik, Ö., 2010. Visual Studio 2008’de Kod Metrikleri ve Kod Analizi (Static and Dynamic Code Analyze) – Bölüm 1. Erişim Tarihi: 15.02.2015. <http://univera-ng.blogspot.com.tr/2010/03/visual-studio-2008de-kod-metrikleri-ve.html>.
- Karayumak, F. 2013. Yazılım Güvenliği Programı. Erişim Tarihi:01.08.2013. <http://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/yazilim-guvenligi-programi.html>.
- Krishna, P.R., De, S.K., 2005. Naive-Bayes Classification using Fuzzy Approach. Intelligent Sensing and Information Processing Conference, 14-17 Eylül, Bangalore, 62-65.
- Kocaoğlu, S., 2010. Arabellek Taşması (Buffer Overflow) Açıklıklarına Karşı Temel Önlemler. Erişim Tarihi:10.08.2014. <http://www.serdarkocaoglu.com.tr/2010/04/arabellek-tasmasi-buffer-overflow.html>.
- Kırbaş, K., 2014. Ldap & Ldap Injection Attack. Erişim Tarihi: 04.04.2014. <http://www.kayhankirbas.com/2011/06/ldap-ldap-injection>.
- Kırdınlı, K., 2010. Visual Studio Kod Metrikleri. Erişim Tarihi: 22.04.2015. <http://koraykirdinli.blogcu.com/visual-studio-kod-metrikleri/8939285>.
- Livshits, V.B., Lam, M.S., 2005. Findings Security Errors in Java Applications Using Lightweight Static Analysis. 14th Usenix Security Symposium. 28 Şubat-3 Mart, Baltimore, 271-286.
- Livshits, V.B., 2005. Findings Security Errors in Java Applications Using Lightweight Static Analysis. In Annual Computer Security Applications Conference. Kasım, Arizona.
- McCabe, T. J., 1976. A Complexity Measure. ", IEEE Transactions on Software Engineering, 2, 308-320.
- McCabe, T J., 2008. Software Quality Metrics to Identify Risk. Erişim Tarihi: 20.05.2015. www.mccabe.com/ppt/SoftwareQualityMetricsToIdentifyRisk.ppt.
- McGraw, G., 2006. Software Security: Building Security In. Addison Wesley Professional, 448, USA.
- Michael, C, C. 2005. Risk-Based and Functional Security Testing. Erişim Tarihi: 16.09.2013. <https://buildsecurityin.us-cert.gov/articles/best-practices/security-testing/risk-base-and-functional-security-testing>.

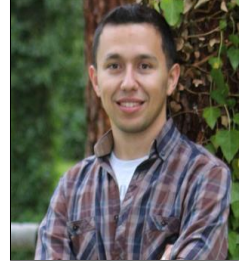
- Muharremoğlu, G., 2012. Oturum Öncesi Tanımlı Oturum Kimliği Çerezi Açığı. Erişim Tarihi: 23.04.2015. <https://www.bilgiguvenligi.gov.tr/web-guvenligi/oturum-oncesi-tanimli-oturum-kimligi-cerezi-acigi.html>.
- Myers, J.G., 2004. The Art of Software Testing. John Wiley & Sons, Inc., Hoboken, 151, Canada.
- Netcom, 2014. Servis Reddetme (Denial of Service, DoS Attack). Erişim Tarihi: 11.04.2014. <http://www.kaynet.net/index.php/coezuemlerimiz/network-guevenlii/itemlist/category/123-servis-reddetme-denial-of-service-dos-attack>.
- Olague, H. M., Etzkorn, L H., and Cox, G. W., 2006. An entropy-based approach to assessing object-oriented software maintainability and degradation-a method and case study, Software Engineering Research and Practice, 442-452.
- Owasp, 2009. Ldap Injection. Erişim Tarihi: 03.03.2014. https://www.owasp.org/index.php/LDAP_injection.
- Owasp, 2013. Top 10 Web Application Security Risk. Erişim Tarihi:10.10.2015. https://www.owasp.org/index.php/Top_10_2013-Top_10.
- Özkan, Y., 2008. Veri Madenciliği Yöntemleri. Papatya Yayıncılık Eğitim, 213, İstanbul.
- Pehlivan, E., 2014. HTTP GET ve POST Yöntemleri. Erişim Tarihi: 15.07.2014. <http://www.yazilimdevi.com/Makaleler-742-http-get-ve-post-yontemleri.aspx>.
- Pesen, M, M. 2013. Bilgi Güvenliği Nedir ve Nasıl Sınıflandırılır. Erişim Tarihi: 27.08.2013. <http://www.sibergah.com/genel/bilgi-guvenligi-nedir-ve-nasil-siniflandirilir/?category=guvenlik/bilgi-guvenligi-nedir-ve-nasil-siniflandirilir/>.
- Php Group, 2015. Öntanımlı Değişkenler. Erişim Tarihi:10.07.2015. <http://php.net/manual/tr/reserved.variables.php>.
- Rahman, U., 2014. Naïve Bayes Sınıflandırma Algoritması. Erişim Tarihi: 30.04.2014. <http://kodcu.com/2014/05/naive-bayes-siniflandirma-algoritmasi/>.
- Ramachandran, M., Nair, T. R. G., Ramachandran, M., Prasad, K., 2010. Software Metrics Evaluation Based on Entropy. Erişim Tarihi:12.06.2015. <http://arxiv.org/abs/1001.3473>.

- Särökaari, N., 2012. Identifying malicious HTTP Requests. Erişim Tarihi: 10.10.2014. <http://pen-testing.sans.org/resources/papers/gwapt/identify-malicious-http-requests-130350>.
- Sarıman, G., Küçüksille, E.U., 2013. Güvenli Yazılım Geliştirme Yaşam Süreci ve Statik Kod Analizi. 6th. International Conference on Information Security and Cryptology. 20-21 Eylül, Ankara, 282-286.
- Satyanarayana, V., Sekhar, M.V.B.C., Static Analysis Tool For Detecting Web Application Vulnerabilities. International Journal of Modern Engineering Research (IJMER), 1(1), 127-133.
- Seyir Defteri, 2013. Denial of Service (DoS) Saldırıları ve Korunma Yöntemleri. Erişim Tarihi: 09.03.2014. [http://bidb.itu.edu.tr/seyrdefteri/blog/2013/09/07/denial-of-service-\(dos\)-sald%C4%B1r%C4%B1lar%C4%B1-ve-korunma-y%C3%B6ntemleri](http://bidb.itu.edu.tr/seyrdefteri/blog/2013/09/07/denial-of-service-(dos)-sald%C4%B1r%C4%B1lar%C4%B1-ve-korunma-y%C3%B6ntemleri).
- Smith, S.A., 2014. Nine Options for Managing Persistent User State in Your ASP.NET Application. Erişim Tarihi: 10.12.2014. http://w.scripts20.com/script/851/27568/Nine_Options_for_Managing_Persistent_User_State_in_Your_AspNet_Application.html.
- Solmaz, R., Günay, M., Alkan, A., 2014. Fonksiyonel Tiroit Hastalığı Tanısında Naive Bayes Sınıflandırıcının Kullanılması. Akademik Bilişim, 5-7 Şubat, Mersin.
- SourceForge, 2011. A static source code analyser for vulnerabilities in PHP scripts. Erişim Tarihi: 13.11.2014. <http://rips-scanner.sourceforge.net>.
- SourceForge, 2015. Code security review tool for C/C++, C#, VB, PHP, Java and PL/SQL. Erişim Tarihi: 05.06.2015. <http://sourceforge.net/projects/visualcodegrepp>.
- Şahinoğlu, M., Sari, M., Kurt, A., Kurnaz, S., Özbek, M., 2013. Yazılım Test Sürecinde Problemler ve Çözüm Önerileri. 7. Ulusal Yazılım Mühendisliği Sempozyumu. 26 Eylül, İzmir, Türkiye.
- Tatlı, İ.E., 2011. Java'da Güvenli Yazılım Geliştirme. Erişim Tarihi: 20.06.2013. http://www.webguvenligi.org/docs/Javada_Guvenli_Yazilim_Gelistirme_OWASPTR.pdf.
- Testingsecurity.com, 2014. Input Validation. Erişim Tarihi: 16.04.2014. <http://www.testingsecurity.com/input-validation>.
- Tiftik, N., Öztarak, H., Ercek, G., Özgün, S., 2007. Sistem/Yazılım Geliştirme Sürecinde Doğrulama Faaliyetleri. 3. Ulusal Yazılım Mühendisliği Sempozyumu. Ankara.

- Tudor, J., 2013. Web Application Vulnerability Statistics. Eriřim Tarihi: 12.10.2014.
http://www.contextis.com/documents/70/Web_Application_Vulnerability_Statistics_Appendix_-_June_2013.pdf.
- Ulakbim, 2009. Web Gvenlięi alıřma Grubu. Eriřim Tarihi:20.09.2013.
<http://csirt.ulakbim.gov.tr/gruplar/zayiflik.uhtml>.
- Urgun, B., 2007. Web Uygulama Gvenlięi Kılavuzu, Ulusal Elektronik Ve Kriptoloji Arařtırma Enstits, 147, Kocaeli.
- nsal, S., 2010. Kod Kalite lmleri ve Sık Kullanılan Metrikler. Eriřim Tarihi: 21.12.2014. <http://nightwalkers.blogspot.com.tr/2010/05/kod-kalite-olcumleri-ve-sk-kullanlan.html>.
- Verisign, 2014. Ddos Saldırısı Nedir?. Eriřim Tarihi: 12.04.2014.
http://www.verisigninc.com/tr_TR/products-and-services/network-intelligence-availability/ddos/ddos-attack/index.xhtml
- Wasserman, M.G., 2008. Techniques and Tools for Engineering Secure Web Applications. University Of California, Doktora Tezi, 103s, California.
- Watson, A. H., McCabe, T.J., 1996. Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. NIST Special Publication, 235-500, Gaithersburg.
- W.O.D, 2013. Ddos Saldırısı Nedir? Eriřim Tarihi: 13.04.2014.
<http://www.bilgebeyin.net/2013/08/ddos-saldrs-nedir.html>. Eriřim Tarihi:13.04.2014.
- Wgerer, W., 2005. A survey of static program analysis techniques. Technische Universit, 16s.
- Widera, M., 2006. Why Testing Matters in Functional Programming. 7th Symposium on Trends in Functional Programming. 19-21 Nisan, University of Nottingham, 55-67.
- Wikipedia, 2009. Dzenli İfade. Eriřim Tarihi: 01.10.2014.
http://tr.wikipedia.org/wiki/D%C3%BCzenli_ifade.
- Wikipedia, 2015. Javascript. Eriřim Tarihi: 05.06.2015.
<http://tr.wikipedia.org/wiki/JavaScript>.
- WhiteHat Security, 2015. Web Site Security Statistics Report. Eriřim Tarihi: 12.08.2015.
<https://info.whitehatsec.com/rs/whitehatsecurity/images/2015-Stats-Report.pdf>.

- Yazıcıoğlu, S., 2007. Querystring Kavramı. Erişim Tarihi:28.11.2014.
<http://www.csharpnedir.com/articles/read/?id=784>.
- Yılmaz, O., 2009. HTML Injection Attack & Security. Erişim Tarihi: 18.07.2013.
http://docs.google.com/Doc?docid=d4w2g9c_14ccmffzd9.
- Yılmaz, O., 2009. Web Uygulamaları Güvenliği ve Web Uygulamalarında Zayıflık Tespiti. Erişim Tarihi: 13.02.2013.
https://docs.google.com/document/d/11VP-XP-WmJmPxrmYSA-zpalkHY9YsKWRlMFFSh0iC_w/edit?pli=1.
- Yılmaz, O., 2013. Yazılımcılar ve Denetimciler için Web Uygulamaları Güvenliği. Erişim Tarihi:11.10.2013.
http://docs.google.com/Doc?id=d4w2g9c_1hpj4w7ft.
- Zainuddin, H. N., Aziz, A. N., 2011. Secure Coding in Software Development. 5th Malaysian Conference in Software Engineering (MySEC), 13-14 Aralık, Johor Bahru, 458-464.
- Zeller, W., Felten, E.W., 2008. Cross-Site Request Forgeries:Exploitation and Prevention. Princeton University, Teknik Rapor, 16s.
- Zheng, J., Williams, L., Nagappan, N., Snipes, W., Hudepohl, J.P., Vouk,M.A., 2006. On the Value of Static Analysis for Fault Detection in Software. IEEE Transactionson Software Engineering, 32(4), 240-253.

ÖZGEÇMİŞ



Adı Soyadı : Güncel SARIMAN

Doğum Yeri ve Yılı : Milas, 1986

Medeni Hali : Evli

Yabancı Dili : İngilizce

E-posta : guncelsariman@mu.edu.tr

Eğitim Durumu

Lise : Milas Anadolu Lisesi, 2004

Lisans : Süleyman Demirel Üniversitesi, Teknik Eğitim Fakültesi,
Bilgisayar Sistemleri Öğretmenliği-2009

Yüksek Lisans : Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü,
Bilgisayar Mühendisliği-2011

Mesleki Deneyim

Süleyman Demirel Üniversitesi Enformatik Bölümü 2009-2010

Süleyman Demirel Üniversitesi Bilgi İşlem Daire Başkanlığı 2010-2012

Muğla Sıtkı Koçman Üniversitesi Bigi İşlem Daire Başkanlığı 2012-

Yayınları

Küçüksille, E.U., Yiğit,T., Güneş, A., Sarıman, G., 2010. A system for the detection and reporting of wireless modem signals, Scientific Research and Essays, 1346-1350.

Sarıman,G., Küçüksille, E.U., 2014. Web Based Educational Tool for Metaheuristic Algorithms. Pamukkale J Eng Sci., 20(2), 46-53.

- Sarıman, G., 2011. Veri Madenciliğinde Kümeleme Teknikleri Üzerine Bir Çalışma:K-Means ve K-Medoids Kümeleme Algoritmalarının Karşılaştırılması, Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 15(3), 192-202.
- Aydin, G., Sarıman, G., 2014. Yabancılar Türkçe Eğitiminde Kelime Öğretimi İçin Web Destekli E-Öğrenme Aracı, 7. Uluslararası Türkçenin Eğitimi - Öğretimi Kurultayı, 19-21 Haziran, Muğla.
- Sarıman, G., Küçüksille, E.U., 2013. Mining Association Rules with Parallel Programming, 3rd World Conference on Innovation and Computer Sciences INSODE-2013, Nisan, Antalya.
- Sarıman, G., Küçüksille, E.U., Güvenli Yazılım Geliştirme Yaşam Süreci ve Statik Kod Analizi, 6th. International Conference on Information Security and Cryptology, Eylül, Ankara.
- Sarıman, G., Tarlaci, F., Karasoy, O., Durmuş, B., 2014. Eduroam ve SMS destekli E-Posta Kullanıcı Yönetim Modeli: MSKU Örneği, Akademik Bilişim 2014, 5-7 Şubat, Mersin.
- Sarıman, G., Karasoy, O., Tarlaci, F., Durmuş, B., 2014. Yazılımlar için web servis destekli bütünleşik hesap yönetimi, Akademik Bilişim 2014, 5-7 Şubat, Mersin.