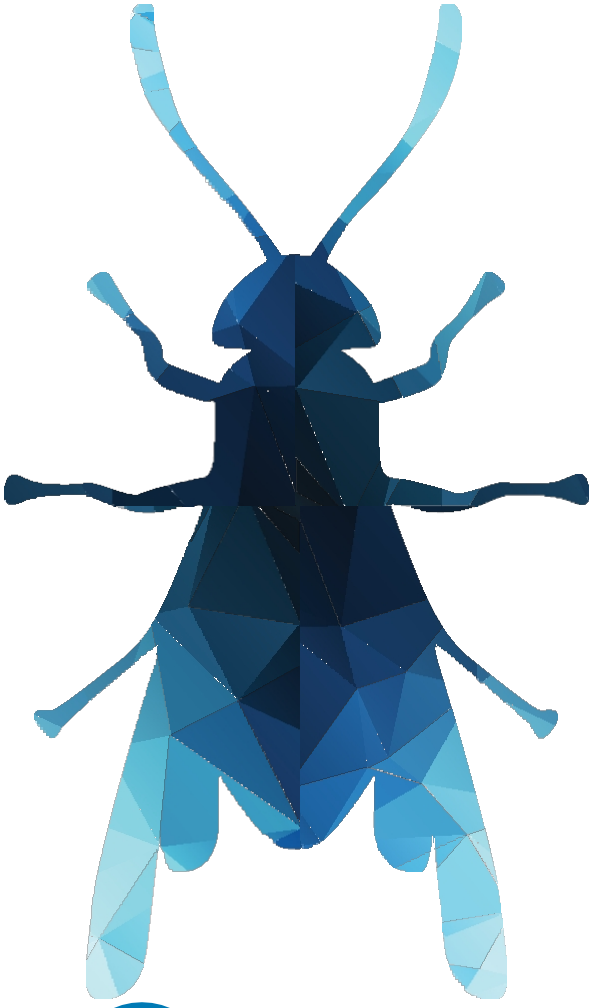
# KOD INCELEME KILAVUZU

2.0

# YAYIN



OWASP
Open Web Application Security Project

Proje liderleri: Larry Conklin ve Gary Robinson

## 1 Giriş



---

## 2



## 3

Güvenli Kod İncelemesi İçin Teknik Referans



Ekler



## 4


1

# 1 ÖNSÖZ

**Eoin Keary tarafından,**
**Uzun Süredir Hizmet Veren OWASP Global Yönetim Kurulu Üyesi**

OWASP Kod İnceleme kılavuzu ilk olarak OWASP Test Kılavuzundan doğmuştur. Başlangıçta kod incelemesi, o zamanlar iyi bir fikir gibi göründüğü için Test Kılavuzu'nda ele alındı. Ancak, güvenlik kodunun gözden geçirilmesi konusu çok büyüktür ve kendi bağımsız kılavuzuna dönüşmüştür.

Kod İnceleme Projesi'ni 2006 yılında başlattım. Bu güncel baskı Nisan 2013'te OWASP Project Reboot girişimi ve Amerika Birleşik Devletleri İç Güvenlik Bakanlığı'ndan alınan bir hibe ile başlatıldı.

OWASP Kod İnceleme ekibi, gerçekten daha sık dışarı çıkması gereken küçük ama yetenekli bir grup gönüllüden oluşmaktadır. Gönüllüler, küçük start-up'lardan dünyanın en büyük yazılım geliştirme organizasyonlarından bazılarına kadar çeşitli kuruluşlarda güvenli kod incelemesinde en iyi uygulamalar konusunda deneyim ve istek sahibidir.

Sistem geliştirme yaşam döngüsünün erken aşamalarında hatalar tespit edildiğinde daha güvenli yazılımların üretilebileceği ve daha uygun maliyetli bir şekilde geliştirilebileceği yaygın bir bilgidir. Yazılım geliştirme yaşam döngüsüne (SDLC) entegre edilmiş uygun bir kod gözden geçirme işlevine sahip kuruluşlar, güvenlik açısından dikkate değer ölçüde daha iyi kodlar üretmektedir. Basitçe söylemek gerekirse "Kendimizi güvenli hale getiremeyiz". Saldırganların bir sistemdeki açıkları bulmak için savunmaya ayrılan zamandan daha fazla zamanı vardır. Kendimizi güvenli hale getirmek için hack'lemek eşit olmayan bir savaş alanı, asimetrik savaş ve kaybedilen bir savaş anlamına gelir.

Zorunlu olarak, bu kılavuz tüm programlama dillerini kapsamamaktadır. Ağırlıklı olarak C#/.NET ve Java'ya odaklanmakla birlikte mümkün olan yerlerde C/C++, PHP ve diğer dilleri de içermektedir. Bununla birlikte, kitapta savunulan teknikler hemen hemen her kod ortamına kolayca uyarlanabilir. Neyse ki (ya da ne yazık ki), web uygulamalarındaki güvenlik açıkları programlama dilleri arasında oldukça tutarlıdır.

**Eoin Keary, Haziran 2017**

## BİRLEŞİK DEVLETLER İÇ GÜVENLİK BAKANLIĞI'NA TEŞEKKÜR

OWASP topluluğu ve Kod İnceleme Kılavuzu proje liderleri, OWASP'a hibe yoluyla sağlanan fonlarla bu kitabın mümkün kılınmasına yardımcı olduğu için Amerika Birleşik Devletleri İç Güvenlik Bakanlığı'na derin teşekkürlerini sunar. OWASP, özgür ve tarafsız uygulama güvenliği için önde gelen kuruluş olmaya devam etmektedir.

Uygulama açıkları yoluyla toplum kuruluşlarına yönelik tehdit ve saldırılarda rahatsız edici bir artış gördük, sadece güçlerimizi birleştirerek ve pişmanlık duymadığımız bilgilerle bu tehditleri geri çevirmeye yardımcı olabiliriz. Dünya artık bir yazılım üzerinde dönüyor ve bu yazılımın güvene layık olması gerekiyor. DHS'ye bu amaca yardımcı olduğu ve paylaştığı için en derin takdir ve teşekkürlerimizi sunarız.

## GERİ BİLDİRİM

OWASP Kod İnceleme ekibi için herhangi bir geri bildiriminiz varsa ve/veya bu Kod İnceleme Kılavuzunda herhangi bir hata veya iyileştirme bulursanız lütfen bizimle iletişime geçin:
**owasp-codereview-project@owasp.org**

# TEŞEKKÜRLER

**Proje Liderleri**

Larry

ConklinGary Robinson

## SÜRÜM 2.0, 2017

**İçerik Katkıda Bulunanlar**
Larry Conklin
Gary Robinson
Johanna Curiel
Eoin Keary
Islam Azeddine Mennouchi
Abbas Naderi
Carlos Pantelides
Michael Hidalgo

**Hakemler**
Alison Shubert
Fernando Galves
Sytze van Koningsveld
Carolyn Cohen
Helen Gao
Jan Masztal
David Li
Lawrence J Timmins
Kwok Cheng
Ken Prole
David D'Amico
Robert Ferris
Lenny Halseth
Kenneth F. Belva

## SÜRÜM 1.0, 2007

**Proje Lideri**
Eoin Keary

**İçerik Katkıda Bulunanlar** Jenelle Chapman Andrew van der Stock Paolo Perego
David Lowry
David Rook
Dinis Cruz
Jeff Williams

**Yorumcular**
Jeff Williams
Rahin Jina

# GİRİŞ

OWASP Kod İnceleme Kılavuzu Projesi'nin ikinci baskısına hoş geldiniz. İkinci baskı, başarılı OWASP Kod İnceleme Kılavuzu'nu güncel tehditler ve karşı önlemlerle güncel hale getiriyor. Bu sürüm ayrıca OWASP topluluklarının güvenli kod inceleme en iyi uygulamalarına ilişkin deneyimlerini yansıtan yeni içerikler de içermektedir.

# İÇİNDEKİLER

Kod İnceleme Kılavuzunun İkinci Baskısı, yazılım geliştiricilere ve yönetime güvenli kod incelemesindeki en iyi uygulamalar ve bunun güvenli yazılım geliştirme yaşam döngüsü (S-SDLC) içinde nasıl kullanılabileceği konusunda tavsiyelerde bulunmak üzere geliştirilmiştir. Kılavuz, okuyucuya güvenli kod incelemesini ve bunun bir şirketin S-SDLC'sine nasıl dahil edilebileceğini tanıtan bölümlerle başlamaktadır. Daha sonra belirli teknik konulara odaklanmakta ve teknik kodu gözden geçirirken bir gözden geçirenin nelere dikkat etmesi gerektiğine dair örnekler sunmaktadır. Kılavuz özellikle şunları kapsamaktadır:

**Genel Bakış**
Bu bölüm okuyucuya güvenli kod incelemesini ve bunun bir geliştirme kuruluşuna getirebileceği avantajları tanıtmaktadır. Güvenli kod inceleme tekniklerine genel bir bakış sunar ve kod incelemenin güvenli kodu analiz etmek için diğer tekniklerle nasıl karşılaştırıldığını açıklar.

**Metodoloji**
Metodoloji bölümü, güvenli gözden geçirme tekniklerinin geliştirme kuruluşlarının S-SDLC'sine nasıl entegre edileceği ve kodu gözden geçiren personelin etkili bir gözden geçirme gerçekleştirmek için doğru bağlama sahip olduklarından nasıl emin olabilecekleri konusunda daha fazla ayrıntıya girmektedir. Konular arasında güvenlik kodu incelemelerine risk tabanlı istihbaratın uygulanması, incelenen uygulamayı anlamak için tehdit modellemesinin kullanılması ve harici iş faktörlerinin güvenli kod incelemesi ihtiyacını nasıl etkileyebileceğinin anlaşılması yer almaktadır.

# KOD GÖZDEN GEÇIRME KILAVUZU NASIL KULLANILIR

Kitabın içeriği ve yapısı özenle tasarlanmıştır. Ayrıca, katkıda bulunulan tüm bölümler titizlikle düzenlenmiş ve yapı ve üslupta tekdüzelik sağlayan birleştirici bir çerçeveye entegre edilmiştir.

**Bu kitap üç farklı bakış açısını tatmin etmek için yazılmıştır.**

**1.** Kod incelemelerinin neden gerekli olduğunu ve günümüz kurumları için güvenli kurumsal yazılım geliştirmede neden en iyi uygulamalara dahil edildiğini anlamak isteyen yönetim ekipleri. Üst yönetim bu kitabın birinci ve ikinci bölümlerini dikkatlice okumalıdır. Güvenli kodlama yapmak kuruluşların yazılım geliştirme yaşam döngüsünün bir parçası olacaksa yönetimin aşağıdaki maddeleri göz önünde bulundurması gerekir:

- Organizasyon proje tahmini kod incelemeleri için zaman ayırıyor mu?

- Yönetim, her proje ve programcı için ilgili kod inceleme ve statik analiz metriklerini takip edebiliyor mu?

- Yönetimin, kod incelemelerinin proje yaşam döngüsünde ne zaman yapılması gerektiğine ve mevcut projelerde hangi değişikliklerin daha önce tamamlanmış kod incelemelerinin gözden geçirilmesini gerektirdiğine karar vermesi gerekir.

**2.** Kurumları için güvenli kurumsal yazılımlar oluşturmaya yardımcı olmak için nelere dikkat etmeleri gerektiği konusunda bol miktarda ampirik eserle kod incelemesinde meslektaşlarına adam gibi geri bildirim vermek isteyen yazılım liderleri. Şunları göz önünde bulundurmalıdırlar:

- Bir akran kod gözden geçiricisi olarak bu kitabı kullanmak için öncelikle ne tür bir kod gözden geçirmesi yapmak istediğinize karar vermelisiniz. Bu kitabın size nasıl yardımcı olabileceğine karar vermenize yardımcı olmak için her bir kod inceleme türünü gözden geçirmek için birkaç dakika harcayalım.

- API/tasarım kodu incelemeleri. Mimari tasarımların nasıl güvenlik açıklarına yol açabileceğini anlamak için bu kitabı kullanın. Ayrıca API üçüncü taraf bir API ise, güvenlik açıklarını önlemek için kodda hangi güvenlik kontrolleri uygulanmaktadır.

- Sürdürülebilirlik kod incelemeleri. Bu tür kod incelemeleri daha çok kuruluşların dahili en iyi kodlama uygulamalarına yöneliktir. Bu kitap, kodun bir bölümü aşırı karmaşıksa, kod gözden geçiren kişinin güvenlik açıkları için hangi koda bakması gerektiğini daha iyi anlamasına yardımcı olabilecek kod metriklerini kapsamaktadır.

- Entegrasyon kodu incelemeleri. Yine bu tür kod incelemeleri daha çok kuruluşların dahili kodlama politikalarına yöneliktir. Projeye entegre edilen kod BT yönetimi tarafından tamamen incelenmiş ve onaylanmış mı? Birçok güvenlik açığı artık güvenli olmayan bağımlılıklar getirebilecek açık kaynak kütüphaneleri kullanılarak uygulanmaktadır.

- Kod incelemelerinin test edilmesi. Programcının, kod yöntemlerinin programcının amaçladığı gibi çalıştığını kanıtlamak için birim testleri oluşturduğu Çevik ve Test Güdümlü tasarım. Bu kod, yazılımı test etmek için bir kılavuz değildir. Kodu gözden geçiren kişi, tüm yöntemlerin uygun istisnalara sahip olduğundan ve kodun güvenli bir şekilde başarısız olduğundan emin olmak için birim test durumlarına dikkat etmek isteyebilir. Mümkünse koddaki her güvenlik kontrolünün uygun birim test durumları vardır.

**3.** Güvenli kod incelemelerinin kuruluşların güvenli yazılım geliştirme yaşam döngüsüne nasıl entegre edildiğine dair güncellenmiş bir kılavuz isteyen güvenli kod incelemecisi. Bu kitap aynı zamanda kod gözden geçirme sürecinde kod gözden geçirme için bir referans kılavuzu olarak da çalışacaktır. Bu kitap, kod gözden geçiren kişinin ihtiyaç duyduğu eksiksiz bir bilgi kaynağı sağlar. İlk olarak kod incelemeleri hakkında bir hikaye olarak, ikinci olarak da bir masaüstü referans rehberi olarak

okunmalıdır.


2

# GÜVENLI KOD INCELEMESI

## Güvenli Kod İncelemesi İçin Teknik Referans

Bu kılavuzda XSS, SQL enjeksiyonu, oturum izleme, kimlik doğrulama, yetkilendirme, günlük kaydı ve bilgi sızıntısı gibi yaygın güvenlik açıkları ve teknik kontroller incelenmekte ve incelemeyi yapan kişiye yol göstermesi için çeşitli dillerde kod örnekleri verilmektedir.

Bu bölüm, çeşitli kontrollerin önemli yönlerini öğrenmek için ve güvenli kod incelemeleri yaparken iş başında bir referans olarak kullanılabilir.

OWASP Top 10 sorunları ile başlıyoruz ve bu sorunların her biri için dikkate alınması gereken teknik hususları açıklıyoruz. Daha sonra OWASP Top 10'a özgü olmayan diğer yaygın uygulama güvenliği sorunlarına geçiyoruz
Güvenli kod incelemesi, sistem geliştirme yaşam döngüsünün erken aşamalarında güvenlik hatalarını tespit etmek için muhtemelen en etkili tekniktir. Otomatik ve manuel sızma testleriyle birlikte kullanıldığında, kod incelemesi bir uygulama güvenlik doğrulama çalışmasının maliyet etkinliğini önemli ölçüde artırabilir.

Bu kılavuz, bir güvenlik kodu incelemesi gerçekleştirmek için bir süreç öngörmemektedir. Daha ziyade, çalışmanın nasıl yapılandırılması ve yürütülmesi gerektiği konusunda rehberlik sağlar. Kılavuz ayrıca belirli güvenlik açıkları için kodun gözden geçirilmesinin mekaniğine odaklanmaktadır.

Manuel güvenli kod incelemesi, güvensiz kodla ilişkili "gerçek risk" hakkında içgörü sağlar. Bu bağlamsal, beyaz kutu yaklaşımı en önemli değerdir. İnsan bir gözden geçiren, koddaki bir hatanın veya güvenlik açığının uygunluğunu anlayabilir. Bağlam, değerlendirilen şeyin insan tarafından anlaşılmasını gerektirir. Uygun bağlam ile hem saldırı olasılığını hem de bir ihlalin iş üzerindeki etkisini hesaba katan ciddi bir risk tahmini yapabiliriz. Güvenlik açıklarının doğru kategorize edilmesi, düzeltme önceliğine ve her şeyi düzeltmekle zaman kaybetmek yerine doğru şeyleri düzeltmeye yardımcı olur.

## 5.1 Kodun Neden Güvenlik Açıkları Vardır?

MITRE, CWE projesinde yaklaşık 1000 farklı türde yazılım zayıflığını kataloglamıştır. Bunların hepsi yazılım geliştiricilerin güvensizliğe yol açan hatalar yapabilmelerinin farklı yollarıdır. Bu zayıflıkların her biri incelikli ve birçoğu da ciddi anlamda zorlayıcıdır. Yazılım geliştiricilere bu zayıflıklar okulda öğretilmez ve çoğu bu sorunlar hakkında iş başında herhangi bir eğitim almaz.

Bu sorunlar son yıllarda çok önemli hale geldi çünkü bağlantıları arttırmaya ve teknoloji ve protokolleri son derece hızlı bir şekilde eklemeye devam ediyoruz. Teknolojiyi icat etme becerisi, onu güvence altına alma becerisini ciddi şekilde geride bıraktı. Bugün kullanılan teknolojilerin birçoğu yeterince (ya da hiç) güvenlik incelemesinden geçmemiştir.

İşletmelerin güvenlik konusuna gereken zamanı ayırmamalarının pek çok nedeni vardır. Genel olarak bu nedenler yazılım pazarındaki temel bir sorundan kaynaklanmaktadır. Yazılım esasen bir kara kutu olduğundan, bir müşterinin iyi kod ile güvensiz kod arasındaki farkı anlaması son derece zordur. Bu görünürlük olmadan satıcılar güvenli kod üretmek için ekstra çaba harcamaya teşvik edilmezler. Bununla birlikte, bilgi güvenliği uzmanları güvenlik kodunun gözden geçirilmesini savunduklarında, güvenlik için daha fazla çaba sarf etmemek için aşağıdaki (haksız) bahanelerle sık sık geri püskürtülürler:

"Asla hacklenmiyoruz (bildiğim kadarıyla), güvenliğe ihtiyacımız yok"

"Uygulamalarımızı koruyan bir güvenlik duvarımız var"

"Çalışanlarımıza uygulamalarımıza saldırmamaları konusunda güveniyoruz"

Son 10 yılda, OWASP Kod İnceleme Projesi'ne dahil olan ekip binlerce uygulama incelemesi gerçekleştirdi ve önemsiz olmayan her uygulamanın güvenlik açıkları olduğunu tespit etti. Kod güvenlik açıkları açısından gözden geçirilmemişse, uygulamada sorun olma olasılığı neredeyse %100'dür.

Yine de kodlarının güvenliği hakkında bilgi sahibi olmamayı tercih eden pek çok kuruluş var. Onlar için, Rumsfeld'in aslında ne bildiğimize dair şifreli açıklamasını düşünün:
"...biliyoruz ki, bilinen bilinen şeyler vardır; bildiğimizi bildiğimiz şeyler vardır. Bilinen bilinmeyenler olduğunu da biliyoruz; yani bilmediğimiz bazı şeyler olduğunu da biliyoruz. Ancak bilinmeyen bilinmeyenler de vardır -- bilmediklerimizi bilmiyoruz."- Donald Rumsfeld

Eğer işletmedeki riskin ölçülmesine dayalı olarak bilinçli kararlar alınıyorsa, bu tamamen desteklenecektir. Ancak, riskler anlaşılmıyorsa, şirket gerektiği gibi özenli davranmıyor ve hem hissedarlara hem de müşterilere karşı sorumsuz davranıyor demektir.

### 5.2 Güvenli Kod İncelemesi Nedir?

Kod incelemesi, uygulamanın özellikleri ve tasarımıyla ilgili güvenlik kusurlarını, kesin kök nedenleriyle birlikte belirlemeyi amaçlar. Uygulamaların artan karmaşıklığı ve yeni teknolojilerin ortaya çıkmasıyla, geleneksel test yöntemi uygulamalarda bulunan tüm güvenlik kusurlarını tespit etmekte başarısız olabilir. Kusurları bulma şansının daha yüksek olması için uygulamanın koduna, harici bileşenlere ve yapılandırmalara hakim olmak gerekir. Uygulama koduna bu şekilde derinlemesine bir dalış, güvenlik kusurlarını önlemek için kullanılabilecek kesin hafifletme tekniklerinin belirlenmesine de yardımcı olur.

Uygun güvenlik ve mantıksal kontrollerin mevcut olduğunu, amaçlandığı gibi çalıştıklarını ve doğru yerlerde çağrıldıklarını doğrulamak için bir uygulamanın kaynak kodunu denetleme sürecidir. Kod incelemesi, uygulamanın bulunduğu ortamda "kendi kendini savunabilecek" şekilde geliştirilmesini sağlamaya yardımcı olmanın bir yoludur.

Güvenli kod incelemesi, bir şirketin uygulama geliştiricilerinin güvenli geliştirme tekniklerini takip ettiğinden emin olmasını sağlar. Genel bir kural olarak, bir sızma testi, uygulama uygun bir güvenlik kodu incelemesinden geçtikten sonra geliştirilen kodla ilgili herhangi bir ek uygulama güvenlik açığı keşfetmemelidir. En azından çok az sayıda sorun keşfedilmelidir.

Tüm güvenlik kodu incelemeleri insan çabası ve teknoloji desteğinin bir kombinasyonudur. Spesifikasyon ölçeğinin bir ucunda elinde metin editörü olan deneyimsiz bir kişi yer alır. Ölçeğin diğer ucunda ise gelişmiş statik analiz (SAST) araçlarına sahip uzman bir güvenlik ekibi yer alır. Ne yazık ki, mevcut uygulama güvenlik araçlarını etkili bir şekilde kullanmak için oldukça ciddi bir uzmanlık seviyesi gerekmektedir. Ayrıca dinamik veri akışını veya iş mantığını da anlamıyorlar. SAST araçları, kapsam ve minimum taban çizgisi belirlemek için harikadır.

Bu görevi yerine getirmek için araçlar kullanılabilir ancak her zaman insan doğrulamasına ihtiyaç duyarlar. Güvenlik kodu incelemesinin kilit taşı olan bağlamı anlamazlar. Araçlar büyük miktarda kodu değerlendirmede ve olası sorunlara işaret etmede iyidir, ancak bir kişinin gerçek bir sorun olup olmadığını, gerçekten istismar edilebilir olup olmadığını belirlemek ve kuruluş için riski hesaplamak için her sonucu doğrulaması gerekir. Otomatik araçların kontrol edemediği önemli kör noktaları doldurmak için insan gözden geçiriciler de gereklidir.

### 5.3 Kod İncelemesi ile Güvenli Kod İncelemesi arasındaki fark nedir?

Yetenek Olgunluk Modeli (CMM), bir yazılım geliştirme kuruluşunun geliştirme süreçlerini ölçmek için yaygın olarak tanınan bir süreç modelidir. Geliştirme süreçlerinin geçici, istikrarsız ve tekrarlanabilir olmadığı 'seviye 1'den, geliştirme süreçlerinin iyi organize edildiği, dokümante edildiği ve sürekli iyileştirildiği 'seviye 5'e kadar uzanır. Bir şirketin geliştirme süreçlerinin başlangıçta 1. seviyede başlayacağı (diğer bir deyişle başlangıç modu) ve kurum olgunlaştıkça ve geliştikçe daha tanımlı, tekrarlanabilir ve genel olarak profesyonel hale geleceği varsayılmaktadır. Kod incelemeleri gerçekleştirme becerisi (bunun henüz güvenli kod incelemesi ile ilgili olmadığını unutmayın), bir kuruluş 2. seviyeye (Tekrarlanabilir) veya 3. seviyeye (Tanımlanmış) ulaştığında devreye girer.

Güvenli Kod İncelemesi, yeniden gözden geçirme sürecinin yapısının, şirket güvenlik standartları gibi güvenlikle ilgili hususları karar verme sürecinin ön saflarına yerleştirdiği standart kod gözden geçirme uygulamasının geliştirilmiş halidir. Bu kararların birçoğu bu belgede açıklanacak ve gözden geçirme sürecinin kod tabanındaki güvenlik risklerini yeterince kapsamasını sağlamaya çalışacaktır, örneğin yüksek riskli kodun daha derinlemesine gözden geçirilmesini sağlamak, gözden geçirenlerin kodu gözden geçirirken doğru güvenlik bağlamına sahip olmalarını sağlamak, gözden geçirenlerin kodu etkili bir şekilde değerlendirmek için gerekli becerilere ve güvenli kodlama bilgisine sahip olmalarını sağlamak.

### 5.4 Güvenli Kaynak Kodu İncelemesinin Ölçeğinin Belirlenmesi?

Güvenli kaynak kodu incelemesinin seviyesi, yazılımın iş veya mevzuat ihtiyaçlarına, uygulamaları yazan yazılım geliştirme organizasyonunun büyüklüğüne ve personelin becerilerine bağlı olarak değişecektir. Performans, ölçeklenebilirlik ve sürdürülebilirlik gibi yazılım geliştirmenin diğer yönlerine benzer şekilde, güvenlik de bir uygulamadaki olgunluğun bir ölçüsüdür. Güvenlik, ticari veya resmi amaçlarla kullanılan her ciddi uygulama veya araçta bulunması gereken fonksiyonel olmayan gerekliliklerden biridir.

Geliştirme ortamı, hobi olarak programlama yapan ve haftalık alışverişlerini visual basic'te takip etmek için bir program yazan (CMM seviye 1) bir kişiden oluşuyorsa, bu programcının kapsamlı düzeyde güvenli kod incelemesi gerçekleştirmek için bu belgedeki tüm tavsiyeleri kullanması pek olası değildir. Diğer uçta ise, yüzlerce uygulama yazan binlerce geliştiriciye sahip büyük bir kuruluş (başarılı olmak istiyorlarsa) tıpkı performans ve ölçeklenebilirliği ciddiye aldıkları gibi güvenliği de çok ciddiye alacaklardır.

Her geliştirme kuruluşunun bu belgedeki tüm konuları takip etme ve uygulama zorunluluğu veya kaynakları yoktur, ancak tüm kuruluşlar geliştirme süreçlerini kendileri için en önemli süreçleri ve teknik tavsiyeleri barındıracak şekilde yazmaya başlayabilmelidir. Bu süreçler daha sonra kuruluş geliştikçe ve olgunlaştıkça güvenli kod inceleme hususlarının daha fazlasını içerecek şekilde genişletilebilir olmalıdır.

Karanlık bir odada 3 kişiden oluşan bir start-up'ta, kodu gönderecek bir 'kod inceleme ekibi' olmayacak, bunun yerine köşedeki, bir zamanlar güvenli kodlama kitabı okuyan ve şimdi onu monitörünü desteklemek için kullanan adam olacaktır.

Orta ölçekli bir şirkette, bazıları güvenlikle ilgilenen ya da uzmanlık alanı güvenlik olan 400 geliştirici olabilir, ancak kuruluşun süreçleri 3 satırlık bir CSS değişikliğini gözden geçirmek için amiral gemisi ürünlerin kimlik doğrulama kodunun yeniden tasarlanmasına verdiği süreyle aynı süreyi verebilir. Buradaki zorluk, işgücünün güvenli kodlama bilgisini (genel olarak) artırmak ve tehdit modellemesi ve güvenli kod incelemesi gibi şeyler yoluyla süreçleri iyileştirmektir.

Binlerce geliştiricisi olan bazı büyük şirketler için S-SDLC'de güvenlik ihtiyacı en üst düzeydedir, ancak süreç verimliliğinin de kârlılık üzerinde etkisi vardır. Örneğin 5.000 geliştiricisi olan büyük bir şirketi ele alalım. Süreçte her bir geliştiricinin bir görevi yerine getirmek için haftada fazladan 15 dakika harcamasına neden olacak bir değişiklik yapılırsa, bu durum şirketin tamamı için her hafta fazladan 1.250 saat anlamına gelir. Bu da sadece yolda kalmak için her yıl fazladan 30 tam zamanlı geliştiriciye ihtiyaç duyulmasıyla sonuçlanır (haftada 40 saat çalışıldığı varsayılırsa). Buradaki zorluk, yaşam döngüsündeki güvenlik değişikliklerinin verimli olmasını ve geliştiricilerin görevlerini yerine getirmelerini engellememesini sağlamaktır.

### Güvenli Kod İncelemesi için İşgücünün Yetiştirilmesi

Kod geliştiricilerin birçoğu güvenlik konusunda bilinçli veya yetenekli olmadığından, bir şirket geliştiriciler arasında akran güvenli kod incelemeleri uygulamalıdır.
Bir iş gücü, güvenli kod inceleme metodolojisini uygulamak için güvenlik becerilerini nasıl kazandırır? Birçok güvenlik olgunluk modeli (örneğin BSIMM veya OpenSAMM), yetenekli geliştiriciler ve yetenekli güvenlik konu uzmanları (KOBİ'ler) olan çekirdek bir güvenlik ekibi kavramını tartışmaktadır. Bir şirketin güvenli bir kod inceleme süreci başlatmasının ilk günlerinde, güvenlik KOBİ'leri yüksek riskli incelemelerin merkezinde yer alacak, deneyim ve bilgilerini kodun risk oluşturabilecek yönlerine işaret etmek için kullanacaklardır.

Çekirdek güvenlik ekibinin yanı sıra, güvenliğe ilgi duyan bir grup geliştirici daha ekip yerel güvenlik KOBİ'leri olarak hareket edebilir ve birçok güvenli kod incelemesinde yer alabilir. Bu uydular (BSIMM'in deyimiyle) teknik konularda çekirdek güvenlik ekibi tarafından yönlendirilecek ve güvenli kodlamanın teşvik edilmesine yardımcı olacaktır.

Zamanla, bir kuruluş çekirdek ve uydu ekiplerinde güvenlik bilgisi oluşturur ve bu da güvenlik bilgisini tüm geliştiricilere yayar, çünkü çoğu kod incelemesinde bir güvenlik SME'si yer alacaktır.

Bunun tüm geliştiricilere verdiği 'iş başında' eğitim çok önemlidir. Bir kuruluş, geliştiricilerini genel güvenlik konularını tanıtacak ve farkındalık yaratacak eğitim kurslarına (sınıf veya CBT) gönderebilir, ancak hiçbir eğitim kursu bir geliştiricinin işiyle %100 ilgili olamaz. Güvenli kod gözden geçirme sürecinde, kodlarını gönderen her geliştirici, gözden geçirme kendi ürettiği kodla ilgili olduğu için tamamen kendileriyle ilgili olan güvenlikle ilgili geri bildirim alacaktır.

Ancak unutulmamalıdır ki, kuruluşun büyüklüğü ne olursa olsun, güvenli kod yeniden gözden geçirmesi yapmanın nedeni daha fazla hata yakalamak ve bunları S-SDLC'de daha erken yakalamaktır. Güvenli kod incelemesi yapmak ve hataları bu şekilde bulmak, hataları testte veya üretimde bulmaya kıyasla daha hızlıdır. 5.000 kişilik bir kuruluş için testte bir hata bulmak, araştırmak, yeniden kodlamak, yeniden gözden geçirmek, yeniden yayınlamak ve yeniden test etmek ne kadar sürer? Ya kod, proje yönetimi ve desteğin sorunu takip etmek ve müşterilerle iletişim kurmak için dahil olacağı üretime geçerse? Belki de haftada 15 dakika bir pazarlık gibi görünecektir.

## 5.5 Kendi Güvenliğimizi Hackleyemeyiz

Sızma testi genellikle zaman içinde yapılan bir kara kutu testidir ve herhangi bir gerilemeyi bulmak için kaynak kodun her sürümünde (veya derlemesinde) tekrarlanmalıdır. Birçok sürekli entegrasyon aracı (örneğin Jenkins/Hudson), otomatik sızma testleri de dahil olmak üzere tekrarlanabilir testlerin bir ürünün oluşturulmuş ve yüklenmiş bir sürümüne karşı çalıştırılmasına izin verir.

Kaynak kodu değiştikçe, bakımı yapılmamış bir sızma testinin bulgularının değeri zamanla azalır. Sızma testlerinin kapsamadığı ancak kod incelemelerinde ele alınabilecek gizlilik, uyumluluk ve kararlılık ve kullanılabilirlik endişeleri de vardır. Örneğin bir bulut ortamındaki veri bilgi sızıntısı bir sızma testi ile keşfedilemeyebilir veya buna izin verilmeyebilir. Bu nedenle sızma testleri cephanelikte önemli bir araç olarak görülmelidir, ancak tek başına ürün yazılımının güvenli olmasını sağlamayacaktır.

Bir yazılım projesindeki güvenlik açıklarını tespit etmenin yaygın yöntemleri şunlardır:

- Kaynak Kod Taraması, bir kaynak kod deposuna veya modüle karşı çalışan otomatik araçları kullanarak, potansiyel olarak güvenlik açıklarına neden olduğu düşünülen dize kalıplarını bulur.

• Otomatik Sızma Testi (kara/gri kutu) sızma testi araçları aracılığıyla, aracın test edilen web sitesiyle ağa kurulduğu ve web sitesi URL'lerine karşı önceden tanımlanmış bir dizi test çalıştırdığı otomatik taramalar.

• Manuel Sızma Testi, yine araçlar kullanılarak, ancak daha karmaşık testler gerçekleştiren bir sızma test uzmanının uzmanlığıyla yapılır.

• Güvenlik konusunda uzman bir kişi ile Güvenli Kod İncelemesi.

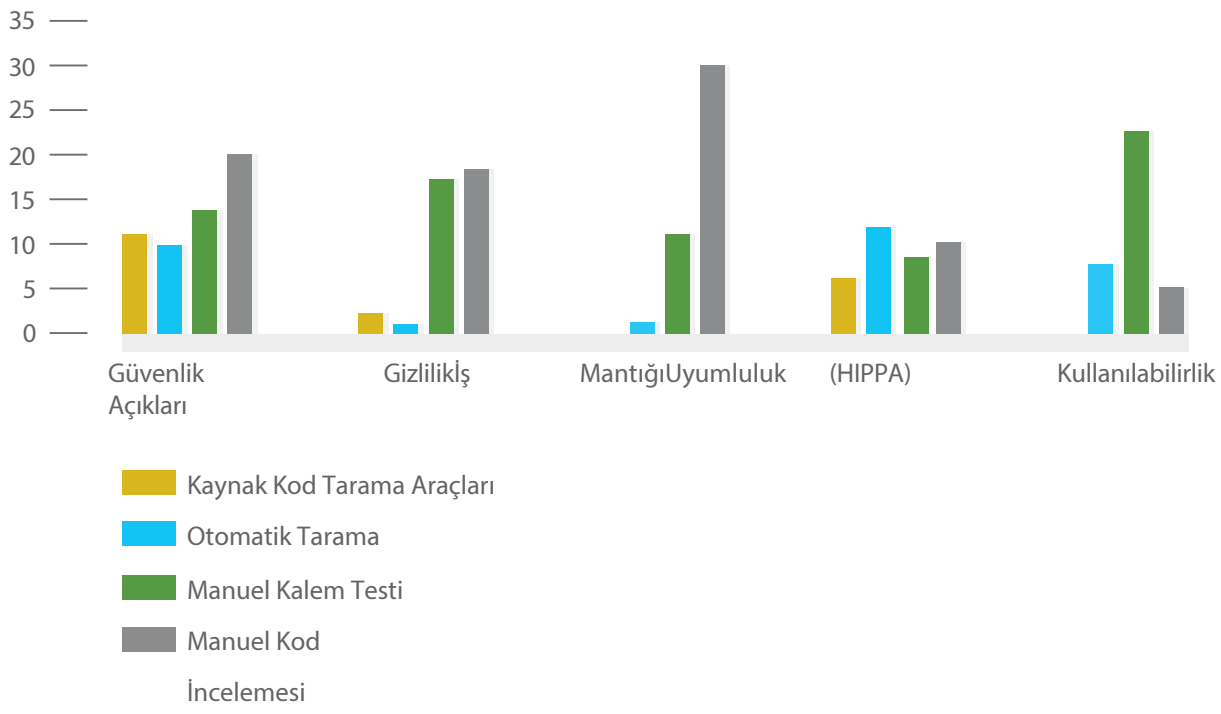
Hiçbir yöntemin bir yazılım projesinin karşılaşabileceği tüm güvenlik açıklarını tespit edemeyeceği, ancak derinlemesine savunma yaklaşımının bilinmeyen sorunların üretim yazılımına dahil edilmesi riskini azaltacağı unutulmamalıdır.

AppSec USA 2015'te yapılan bir ankette katılımcılar hangi güvenlik yönteminin en etkili olduğunu değerlendirdi Buldum:

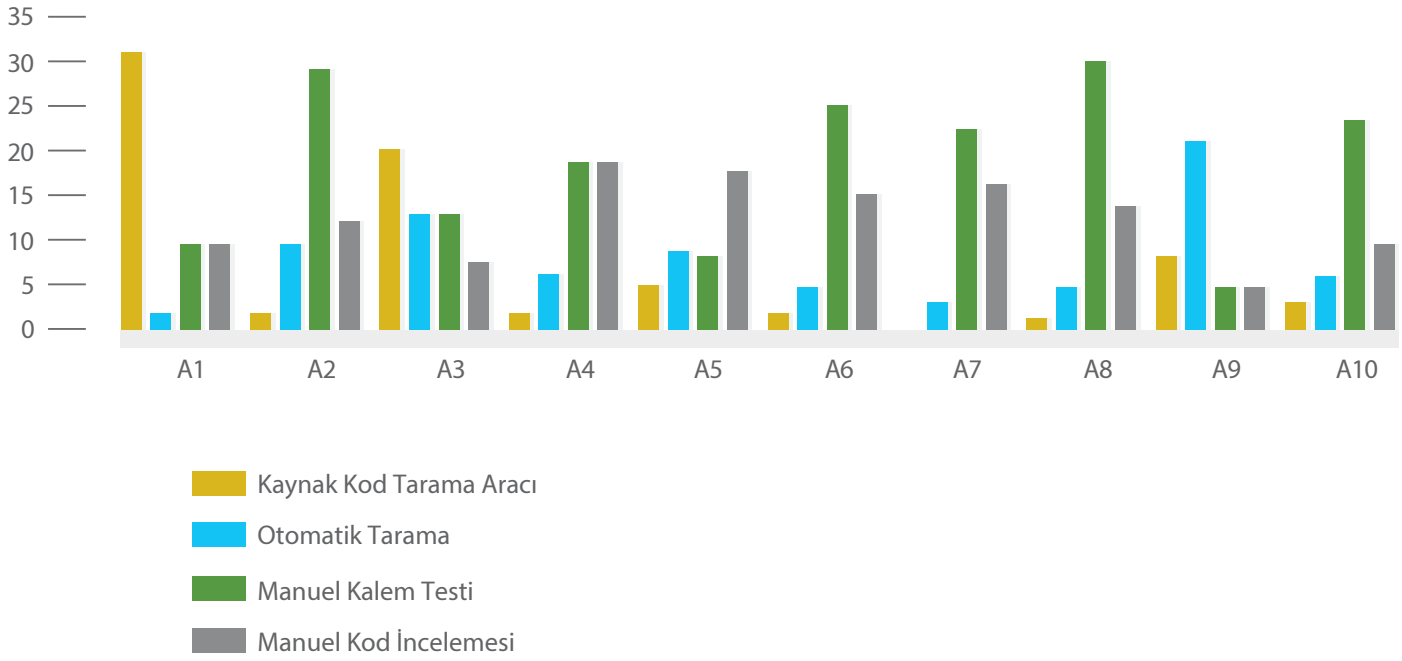
**1)** Genel güvenlik açıkları
**2)** Gizlilik sorunları
**3)** İş mantığı hataları
**4)** Uyumluluk sorunları (HIPPA, PCI vb. gibi)
**5)** Kullanılabilirlik sorunları

Sonuçlar şekil 1'de gösterilmektedir.

**Şekil 1:** Tespit yöntemlerini genel güvenlik açığı türleriyle ilişkilendiren



**Şekil 2:** Tespit yöntemlerini OWASP Top 10 güvenlik açığı türleriyle ilişkilendiren

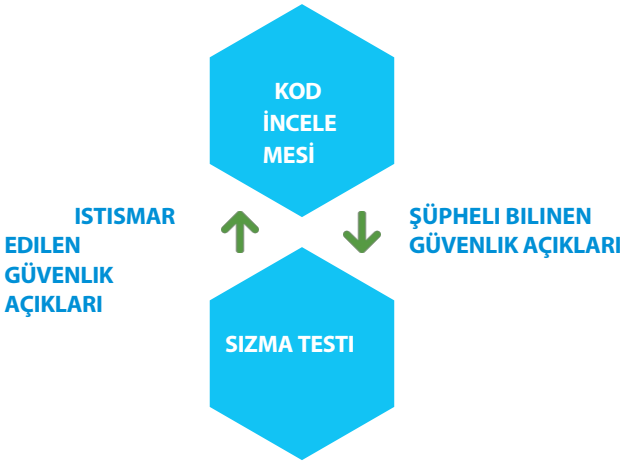



Bu anketler, manuel kod incelemesinin bir şirketin güvenli yaşam döngüsünün bir bileşeni olması gerektiğini, çünkü çoğu durumda güvenlik sorunlarını tespit etmek için diğer yöntemler kadar iyi veya daha iyi olduğunu göstermektedir.

### 5.6 Kaynak Kod İncelemesi ve Sızma Testinin Birleştirilmesi

"360 inceleme" terimi, bir kaynak kodu incelemesinin sonuçlarının bir sızma testini planlamak ve yürütmek için kullanıldığı ve sızma testinin sonuçlarının da ek kaynak kodu incelemesini bilgilendirmek için kullanıldığı bir yaklaşımı ifade eder.

**Şekil 3:** Kod İnceleme ve Sızma Testi Etkileşimleri



Kod incelemesinden iç kod yapısını bilmek ve bu bilgiyi test senaryoları ve kötüye kullanım senaryoları oluşturmak için kullanmak beyaz kutu testi olarak bilinir (aynı zamanda açık kutu ve cam kutu testi olarak da adlandırılır). Bu yaklaşım daha verimli bir sızma testine yol açabilir, çünkü testler şüphelenilen ve hatta bilinen güvenlik açıklarına odaklanabilir. Web uygulamasında kullanılan belirli çerçeveler, kütüphaneler ve diller hakkında bilgi sahibi olan sızma testi, bu çerçevelerde, kütüphanelerde ve dillerde var olduğu bilinen zayıflıklara odaklanabilir.

Beyaz kutu sızma testi, kod incelemesi yoluyla keşfedilen bir güvenlik açığının oluşturduğu gerçek riski belirlemek için de kullanılabilir. Kod incelemesi sırasında bulunan bir güvenlik açığı, kodu gözden geçiren(ler)in koruyucu bir önlemi (örneğin girdi doğrulama) dikkate almaması nedeniyle pen- etrasyon testi sırasında istismar edilemez hale gelebilir. Bu durumda güvenlik açığı gerçek olsa da, maruz kalma eksikliği nedeniyle gerçek risk daha düşük olabilir. Bununla birlikte, koruyucu önlemin gelecekte değiştirilmesi ve dolayısıyla güvenlik açığının ortaya çıkması durumunda sızma testini eklemenin hala bir avantajı vardır.

Beyaz kutu sızma testi (güvenli kod incelemesine dayalı) sırasında istismar edilen güvenlik açıkları kesinlikle gerçek olsa da, bu güvenlik açıklarının gerçek riski dikkatle analiz edilmelidir. Bir saldırganın hedef web uygulamasının kaynak koduna erişmesi ve geliştiricilerinden tavsiye alması gerçekçi değildir. Bu nedenle, dışarıdan bir saldırganın beyaz kutu sızma test cihazı tarafından bulunan güvenlik açıklarından yararlanma riski muhtemelen düşüktür. Ancak, web uygulaması kuruluşu içeriden bilgi sahibi saldırganların (eski çalışanlar veya mevcut çalışanlar veya yüklenicilerle gizli anlaşma) riskiyle ilgileniyorsa, gerçek dünya riski de aynı derecede yüksek olabilir.

Sızma testinin sonuçları daha sonra kod incelemesi için ek alanları hedeflemek üzere kullanılabilir. Testte istismar edilen belirli bir güvenlik açığını ele almanın yanı sıra, testte açıkça istismar edilmemiş olsa bile, aynı güvenlik açığı sınıfının mevcut olduğu ek yerleri aramak iyi bir uygulamadır. Örneğin, uygulamanın bir alanında çıktı kodlaması kullanılmıyorsa ve sızma testi bunu istismar ettiyse, çıktı kodlamasının uygulamanın başka bir yerinde de kullanılmaması oldukça olasıdır.

## 5.7 Kod İncelemesinin Geliştirme Uygulamalarına Örtük Avantajları

Kod incelemesini bir şirketin geliştirme süreçlerine entegre etmek, kod incelemelerini gerçekleştirmek için kullanılan süreçlere ve araçlara, bu verilerin ne kadar iyi yedeklendiğine ve bu araçların nasıl kullanıldığına bağlı olarak birçok fayda sağlayabilir. Geliştiricileri bir odaya toplayıp bir projektörde kod gösterme ve inceleme sonuçlarını basılı bir kopyaya kaydetme günleri çoktan geride kaldı, bugün kod incelemesini daha verimli hale getirmek ve inceleme kayıtlarını ve kararlarını takip etmek için birçok araç mevcut. Kod gözden geçirme süreci doğru yapılandırıldığında, kod gözden geçirme eylemi verimli olabilir ve herhangi bir kuruluşa eğitimsel, denetlenebilir ve tarihsel faydalar sağlayabilir. Bu bölümde, bir kod gözden geçirme prosedürünün geliştirme organizasyonuna katabileceği faydaların bir listesi sunulmaktadır.

**Tarihsel bir kayıt sağlar**

Herhangi bir geliştirici bir şirkete katılmışsa veya bir şirket içinde ekip değiştirmişse ve yıllar önce yazılmış bir kod parçasının bakımını yapmak veya geliştirmek zorunda kalmışsa, en büyük hayal kırıklıklarından biri yeni geliştiricinin eski kod hakkında sahip olduğu bağlamın eksikliği olabilir. Kod dokümantasyonu konusunda hem kod içinde (yorumlar) hem de kod dışında (tasarım ve işlev belgeleri, wiki'ler vb.) çeşitli görüşler mevcuttur. Görüşler, sıfır dokümantasyon toleransından, dokümantasyonun boyutunun kod modülünün boyutunu çok aştığı NASA seviyesine yakın dokümantasyona kadar değişmektedir.

Bir kod incelemesi sırasında meydana gelen tartışmaların çoğu, eğer kaydedilirse, modül bakımcılarına ve yeni programcılara değerli bilgiler (metin) sağlayacaktır. Yazarın modülü bazı tasarım kararlarıyla birlikte tanımlamasından, her bir gözden geçirenin neden bir SQL sorgusunun yeniden yapılandırılması ya da bir algoritmanın değiştirilmesi gerektiğini düşündüklerini belirten yorumlarına kadar, gözden geçirenlerin gözleri önünde, gözden geçirme toplantılarına dahil olmayan modülün gelecekteki kodlayıcıları tarafından

kullanılabilecek bir geliştirme hikayesi ortaya çıkmaktadır.

Bu gözden geçirme tartışmalarının bir gözden geçirme aracında otomatik olarak yakalanması ve ileride başvurulmak üzere saklanması, geliştirme organizasyonuna modül üzerindeki değişikliklerin bir geçmişini sunacak ve daha sonra yeni geliştiriciler tarafından sorgulanabilecektir. Bu tartışmalar aynı zamanda mimari/fonksiyonel/tasarım/test spesifikasyonlarına, hata veya geliştirme numaralarına bağlantılar içerebilir.

**Değişikliğin test edildiğinin doğrulanması**
Bir geliştirici kodu depoya göndermek üzereyken, şirket kodu yeterince test ettiğini nasıl bilebilir? Değiştirilen koda karşı çalıştırdıkları testlerin (manuel veya otomatik) bir açıklamasını eklemek, gözden geçirenlere (ve yönetime) değişikliğin çalışacağı ve herhangi bir gerilemeye neden olmayacağı konusunda güven verebilir. Ayrıca yazar, değişikliklerine karşı çalıştırdığı testleri bildirerek, gözden geçirenlerin testleri incelemesine ve yazar tarafından gözden kaçırılmış olabilecek daha fazla test önermesine izin verir.

Otomatik birim veya bileşen testlerinin mevcut olduğu bir geliştirme senaryosunda, kodlama yönergeleri geliştiricinin bu birim/bileşen testlerini kod incelemesine dahil etmesini gerektirebilir. Bu da yine bu ortamdaki gözden geçirenlerin doğru birim/bileşen testlerinin ortama dahil edileceğinden emin olmalarını ve sürekli entegrasyon döngülerinin kalitesini korumalarını sağlar.

**Genç geliştiriciler için kodlama eğitimi**
Bir çalışan bir dilin temellerini öğrendikten ve en iyi uygulamalar kitabından birkaçını okuduktan sonra, daha fazlasını öğrenmek için iş başında nasıl iyi beceriler edinebilir? Eşli kodlama (nadiren gerçekleşen ve asla uygun maliyetli olmayan) ve eğitim oturumlarının (kodlama üzerine kahverengi çanta oturumları, teknoloji konuşmaları vb.) yanı sıra, bir kod incelemesi sırasında tartışılan tasarım ve kod kararları, genç geliştiriciler için bir öğrenme deneyimi olabilir. Birçok deneyimli geliştirici bunun iki yönlü bir yol olduğunu, yeni geliştiricilerin eski geliştiricilerin öğrenebileceği yeni fikirler veya püf noktaları ile gelebileceğini kabul eder. Deneyim ve fikirlerin bu çapraz tozlaşması bir geliştirme organizasyonu için sadece faydalı olabilir.

**Kod tabanına aşinalık**
Yeni bir özellik geliştirildiğinde, genellikle ana kod tabanı ile entegre edilir ve burada kod incelemesi, daha geniş ekibin yeni özellik ve kodunun ürünü nasıl etkileyeceği hakkında bilgi edinmesi için bir kanal olabilir. Bu, ayrı ekiplerin aynı küçük işlevsellik parçasını kodladığı işlevsel yinelemeyi önlemeye yardımcı olur.

Bu aynı zamanda silo halinde ekiplerin bulunduğu geliştirme ortamları için de geçerlidir. Burada kod inceleme yazarı diğer ekiplere ulaşarak onların görüşlerini alabilir ve diğer ekiplerin kendi modüllerini incelemesine izin verebilir ve böylece herkes şirketin kod tabanı hakkında biraz daha fazla bilgi sahibi olur.

**Entegrasyon çatışmaları için ön uyarı**
Yoğun bir kod tabanında, birden fazla geliştiricinin aynı modülü etkileyen kod yazabileceği zamanlar (özellikle temel kod modüllerinde) olacaktır. Pek çok kişi kodu kesip testleri çalıştırdıktan sonra başka bir değişikliğin işlevselliği değiştirdiğini ve yazarın değişikliklerinin bazı yönlerini yeniden kodlamasını ve yeniden test etmesini gerektirdiğini keşfetme deneyimine sahip olmuştur. Kod incelemeleri aracılığıyla yaklaşan değişiklikler hakkında bilgi vermek, bir geliştiricinin bir değişikliğin yaklaşan taahhütlerini etkilemek üzere olduğunu öğrenmesi için daha büyük bir şans verir ve geliştirme zaman çizelgeleri vb. buna göre güncellenebilir.

## Güvenli Kodlama Yönergeleri Temas Noktası

Birçok geliştirme ortamında yeni kodun uyması gereken kodlama yönergeleri vardır. Kodlama yönergeleri birçok şekilde olabilir. Güvenlik yönergelerinin özellikle ilgili bir temas noktası olabileceğini belirtmek gerekir

Ne yazık ki güvenli kodlama konuları geliştirme ekibinin sadece bir alt kümesi tarafından anlaşılmaktadır. Bu nedenle, farklı teknik uzmanlıklara sahip ekipleri kod incelemelerine dahil etmek mümkün olabilir,
Örneğin, güvenlik ekibinden biri (ya da köşedeki tüm güvenlik konularını bilen kişi) kodu kendi bakış açısından kontrol etmesi için teknik konu uzmanı olarak incelemeye davet edilebilir. OWASP ilk 10 yönergesi burada uygulanabilir.

### 5.8 Güvenli Kod İncelemesinin Teknik Yönleri

Güvenlik kodu incelemeleri, incelenen uygulamaya çok özeldir. Yürütme akışının güvensiz bir şekilde sonlandırılması, senkronizasyon hataları gibi yeni veya uygulamanın kod uygulamasına özgü bazı kusurları vurgulayabilirler. Bu kusurlar ancak uygulama kod akışını ve mantığını anladığımızda ortaya çıkarılabilir. Dolayısıyla, güvenlik kodu incelemesi sadece bilinmeyen güvensiz kod kalıpları için kodu taramaktan ibaret değildir, aynı zamanda uygulamanın kod uygulamasını anlamayı ve buna özgü kusurları sıralamayı da içerir.

İncelenen uygulama, merkezi bir kara liste, girdi doğrulama vb. gibi bazı güvenlik kontrolleri ile tasarlanmış olabilir. Bu güvenlik kontrolleri, hatasız olup olmadıklarını belirlemek için dikkatle incelenmelidir. Kontrolün uygulanmasına göre, saldırının doğası veya onu atlamak için kullanılabilecek herhangi bir özel saldırı aracı analiz edilmelidir. Mevcut güvenlik kontrolündeki zayıflıkların sıralanması, güvenlik kodu incelemelerinin bir diğer önemli yönüdür.

Uygulamada güvenlik kusurlarının ortaya çıkmasının, girdi doğrulama eksikliği veya parametre yanlış kullanımı gibi çeşitli nedenleri vardır. Bir kod gözden geçirme sürecinde, kusurların kesin kök nedeni ortaya çıkarılır ve tüm veri akışı izlenir. 'Kaynaktan lavaboya analiz' terimi, uygulamaya gelen tüm olası girdilerin (kaynak) ve bunların uygulama tarafından nasıl işlendiğinin (lavabo) belirlenmesi anlamına gelir. Lavabo, dinamik bir SQL sorgusu, bir günlük yazarı veya bir istemci cihazına verilen bir yanıt gibi güvensiz bir kod modeli olabilir.

Kaynağın bir kullanıcı girdisi olduğu bir senaryo düşünün. Bu girdi uygulamanın farklı sınıflarından/bileşenlerinden geçerek nihayetinde birleştirilmiş bir SQL sorgusuna (bir sink) düşer ve bu yol üzerinde uygun bir doğrulama uygulanmaz. Bu durumda uygulama, kaynaktan lavaboya analizi ile tespit edildiği üzere SQL enjeksiyon saldırısına karşı savunmasız olacaktır. Böyle bir analiz, hangi savunmasız girdilerin uygulamada bir istismar olasılığına yol açabileceğini anlamaya yardımcı olur.

Bir kusur tespit edildiğinde, gözden geçiren kişi uygulamada mevcut olan tüm olası örnekleri sıralamalıdır. Bu, bir kod değişikliği ile başlatılan bir kod incelemesi değil, bir kusurun keşfedilmesi ve kaynakların bu kusurun ürünün diğer bölümlerinde de olup olmadığını bulmak için harcanmasına dayalı olarak yönetim tarafından başlatılan bir kod taraması olacaktır. Örneğin, bir uygulama, scriptlet'lerin 'response.write' yöntemi gibi güvenli olmayan görüntüleme yöntemlerinde doğrulanmamış girdilerin kullanılması nedeniyle XSS güvenlik açığına karşı savunmasız olabilir.

### 5.9 Kod İncelemeleri ve Mevzuata Uygunluk

Yazılımlarının ve verilerinin bütünlüğünü, gizliliğini ve kullanılabilirliğini koruma sorumluluğuna sahip birçok kuruluşun yasal uyumluluğu karşılaması gerekir. Bu uyumluluk genellikle kuruluş tarafından atılan gönüllü bir adım olmaktan ziyade zorunludur.

**Uyum düzenlemeleri şunları içerir:**

- PCI (Ödeme Kartı Endüstrisi) standartları
- Merkez bankası düzenlemeleri

- Denetim hedefleri
- HIPPA

Uyumluluk, yazılım güvenliği geliştirme yaşam döngüsünün ayrılmaz bir parçasıdır ve kod incelemesi uyumluluğun önemli bir parçasıdır, çünkü birçok kural belirli düzenlemelere uymak için kod incelemelerinin yürütülmesinde ısrar eder.

Uyumluluk kurallarını karşılayan uygun kod incelemeleri gerçekleştirmek için onaylanmış bir yöntem kullanmak zorunludur. PCI gibi uyumluluk gereksinimleri, özellikle gereksinim 6: "Güvenli sistemler geliştirin ve sürdürün", Kasım 2013'ten beri mevcut olan PCI-DSS 3.0 ise yazılım geliştirme ve koddaki güvenlik açıklarını belirleme için geçerli olan bir dizi gereksinimi ortaya koymaktadır. Ödeme Kartı Endüstrisi Veri Güvenliği Standardı (PCI-DSS) Haziran 2005'te kredi kartı ödemelerini işleyen şirketler için zorunlu bir uyum adımı haline gelmiştir. Özel kod üzerinde kod incelemeleri gerçekleştirmek, standardın ilk versiyonundan bu yana bir gereklilik olmuştur.

PCI standardı, güvenli uygulama geliştirmeyle ilgili çeşitli noktalar içerir, ancak bu kılavuz yalnızca kod incelemelerini zorunlu kılan noktalara odaklanacaktır. Kod incelemeleri ile ilgili tüm noktalar gereklilik 6 "Güvenli sistemler ve uygulamalar geliştirin ve sürdürün" bölümünde bulunabilir.

### 5.10 Kod İncelemesiyle İlgili PCI-DSS Gereklilikleri

Özellikle, gereklilik 6.3.2 özel kodun kod incelemesini zorunlu kılmaktadır. Herhangi bir potansiyel kodlama açığını belirlemek için (manuel veya otomatik süreçler kullanarak) üretime veya müşterilere yeniden kiralamadan önce özel kodun gözden geçirilmesi en azından aşağıdakileri içerir:

- Kod değişiklikleri, ilk kod yazarı dışındaki kişiler tarafından ve kod gözden geçirme teknikleri ve güvenli kodlama uygulamaları hakkında bilgi sahibi olan kişiler tarafından gözden geçirilir.

- Kod incelemeleri, kodun güvenli kodlama yönergelerine göre geliştirilmesini sağlar

- Serbest bırakılmadan önce uygun düzeltmeler yapılır.

- Kod inceleme sonuçları yayınlanmadan önce yönetim tarafından gözden geçirilir ve onaylanır.

**Gereksinim 6.5, yazılım geliştirme süreçlerindeki yaygın kodlama zafiyetlerini aşağıdaki şekilde ele almaktadır:**

- Geliştiricileri, yaygın kodlama açıklarından nasıl kaçınılacağı ve hassas verilerin bellekte nasıl işlendiğinin anlaşılması dahil olmak üzere güvenli kodlama teknikleri konusunda eğitin.

- Güvenli kodlama yönergelerine dayalı uygulamalar geliştirin.

PCI Konseyi birinci seçeneği kod incelemesi yapan dahili kaynakları da kapsayacak şekilde genişletmiştir. Bu, dahili bir kod incelemesine ağırlık katmıştır ve bu sürecin doğru bir şekilde gerçekleştirilmesini sağlamak için ek bir neden sağlamalıdır.

Ödeme Uygulaması Veri Güvenliği Standardı (PA-DSS), PCI-DSS'ye benzer bir dizi kural ve gerekliliktir. Ancak, PA-DSS özellikle yazılım satıcıları ve yetkilendirme veya mutabakatın bir parçası olarak kart sahibi verilerini saklayan, işleyen veya ileten ödeme uygulamaları geliştiren ve bu ödeme uygulamalarının üçüncü taraflara satıldığı, dağıtıldığı veya lisanslandığı diğer kişiler için geçerlidir.

**Kod İncelemesiyle İlgili PA-DSS Gereklilikleri**
Kod incelemesine ilişkin gereklilikler de uygulanmaktadır çünkü bunlar PA-DSS'den 5. gereklilikte türetilmiştir (PCI, 2010):

5.2 Tüm ödeme uygulamalarını (dahili ve harici ve ürüne web üzerinden idari erişim dahil) güvenli kodlama yönergelerine göre geliştirin.

5.1.4 Herhangi bir önemli değişiklikten sonra müşterilere yayınlanmadan önce, olası kodlama güvenlik açıklarını belirlemek için ödeme uygulaması kodunun gözden geçirilmesi.

**Not:** Kod incelemelerine yönelik bu gereklilik, sistem geliştirme yaşam döngüsünün bir parçası olarak tüm ödeme uygulaması bileşenleri (hem dahili hem de kamuya yönelik web uygulamaları) için geçerlidir. Kod incelemeleri bilgili dahili personel veya üçüncü taraflarca gerçekleştirilebilir.

# METODOLOJİ

Kod incelemesi, bilgisayar kaynak kodunun sistematik olarak incelenmesidir ve incelemeler çeşitli şekillerde yapılır ve her kuruluşun S-SDLC'sinin çeşitli aşamalarında gerçekleştirilebilir. Bu kitap her kuruluşa kendi kuruluşlarında kod incelemelerini nasıl uygulayacaklarını anlatmaya çalışmaz, ancak bu bölüm genel terimlerle ve gayri resmi gözden geçirmelerden, resmi incelemelerden veya Araç-assist-ed kod incelemelerinden kod incelemeleri yapma metodolojisini ele alır.

## 6.1 Bir Kod İnceleme Süreci Geliştirirken Dikkate Alınması Gereken Faktörler

Bir güvenlik kodu incelemesi gerçekleştirmeyi planlarken, her kod incelemesi kendi bağlamına özgü olduğu için dikkate alınması gereken birden fazla faktör vardır. Bu bölümde tartışılan unsurlara ek olarak, analizi etkileyen teknik veya işle ilgili faktörler (son tarihler ve kaynaklar gibi iş kararları) dikkate alınmalıdır, çünkü bu faktörler sonuçta kod incelemesinin gidişatını ve yürütmenin en etkili yolunu belirleyebilir.

### Riskler

Her şeyi %100 güvenli hale getirmek mümkün değildir, bu nedenle risk temelli bir yaklaşımla hangi özelliklerin ve bileşenlerin güvenli bir şekilde gözden geçirilmesi gerektiğine öncelik vermek önemlidir. Bu proje, tasarım güvenliğinin bazı hayati alanlarını vurgularken, eş programcılar bir depoya gönderilen tüm kodları gözden geçirmelidir, ancak tüm kodlar güvenli bir kod incelemesinin dikkatini ve incelemesini almayacaktır.

### Amaç ve Bağlam

Bilgisayar programlarının farklı amaçları vardır ve sonuç olarak güvenlik derecesi uygulanan işlevselliğe bağlı olarak değişecektir. Bir ödeme web uygulaması, bir promosyon web sitesinden daha yüksek güvenlik standartlarına sahip olacaktır. İşletmenin neyi korumak istediğini aklınızda tutun. Bir ödeme uygulaması söz konusu olduğunda, kredi kartları gibi veriler en yüksek önceliğe sahip olacaktır, ancak bir tanıtım web sitesi söz konusu olduğunda, korunması gereken en önemli şeylerden biri web sunucularına bağlantı kimlik bilgileri olacaktır. Bu, bağlamı risk temelli bir yaklaşıma yerleştirmenin başka bir yoludur. Güvenlik incelemesini yürüten kişiler bu önceliklerin farkında olmalıdır.

### Kod Satırları

İş miktarının bir göstergesi de gözden geçirilmesi gereken kod satırlarının sayısıdır. Visual Studio veya Eclipse gibi IDE'ler (Integrated De- velopment Environments) kod satırlarının miktarının hesaplanmasını sağlayan özellikler içerir veya Unix/Linux'ta satırları sayabilen 'wc' gibi basit araçlar vardır. Nesne yönelimli dillerde yazılan programlar sınıflara ayrılır ve her sınıf bir sayfa koda eşdeğerdir. Genel olarak satır numaraları, düzeltilmesi gereken kodun tam yerini belirlemeye yardımcı olur ve bir geliştirici tarafından yapılan düzeltmeleri gözden geçirirken (bir kod deposundaki geçmiş gibi) çok yararlıdır. Bir program ne kadar çok kod satırı içeriyorsa, kodda hata bulunma olasılığı da o kadar yüksektir.

### Programlama dili

Yazılı güvenli dillerde (C# veya Java gibi) yazılan programlar, C ve C++ gibi diğerlerine göre buf- fer taşmaları gibi belirli güvenlik hatalarına karşı daha az savunmasızdır. Kod incelemesi yapılırken, dilin türü beklenen hata türlerini belirleyecektir. Tipik olarak yazılım evleri programcılarının deneyimli olduğu birkaç dile yönelir, ancak geliştiricinin yeni bir dilde yeni kod oluşturmaya karar verdiğinde yönetim, kurum içi deneyim eksikliği nedeniyle bu kodun güvenli bir şekilde gözden geçirilme riskinin arttığının farkında olmalıdır. Bu kılavuz boyunca bölümler, incelenecek belirli programlama dili koduyla ilgili en yaygın sorunları açıklamaktadır, koddaki belirli güvenlik sorunlarını tespit etmek için bunu bir referans olarak kullanın.

**Kaynaklar, Zaman ve Son Tarihler**
Her zaman olduğu gibi, bu temel bir faktördür. Karmaşık bir program için uygun bir kod incelemesi daha uzun sürecek ve basit bir programa göre daha yüksek analiz becerileri gerektirecektir. Kaynakların uygun şekilde sağlanmaması durumunda ortaya çıkan riskler daha yüksektir. Bir inceleme gerçekleştirirken bunun açıkça değerlendirildiğinden emin olun.

### 6.2 Kod İncelemelerinin S-SDLC'ye Entegre Edilmesi

Kod incelemeleri her resmi Güvenli Yazılım Geliştirme Yaşam Döngüsünde (S-SDLC) mevcuttur, ancak kod incelemeleri de resmiyet düzeyleri açısından büyük farklılıklar gösterir. Konuyu daha da karmaşık hale getirmek için, kod incelemeleri amaca ve kodu inceleyen kişinin güvenlik, uyumluluk, programlama stili vb. gibi konularda ne aradığına bağlı olarak değişir. S-SDLC (XP, Agile, RAD, BSIMM, CMMI, Microsoft ALM) boyunca bir uygulama güvenliği KOBİ'sinin dahil olması gereken noktalar vardır. Güvenli kod incelemelerini bir S-SLDC'ye entegre etme fikri kulağa ürkütücü gelebilir çünkü zaten bütçe ve zaman açısından kısıtlı olan bir projeye yeni bir karmaşıklık katmanı ya da ek maliyet ve zaman eklenecektir. Ancak maliyet etkin olduğu kanıtlanmıştır ve statik analizörlerin sağlayamayacağı ek bir güvenlik düzeyi sağlar.

Bazı sektörlerde bir şirketin S-SDLC'sinde güvenli geliştirmeler yapma dürtüsü yalnızca daha iyi kod üretme arzusundan kaynaklanmayabilir, bu sektörlerde yazılım yazarken gereken özeni göstermeyi gerektiren yönetmelikler ve yasalar vardır (örneğin devlet ve finans sektörleri) ve S-SDLC'sini güvenli hale getirmeye çalışmayan bir şirkete kesilecek cezalar, geliştirme yaşam döngüsüne güvenlik eklemenin maliyetinden çok daha fazla olacaktır.

Güvenli kod incelemelerini S-SDLC'ye entegre ederken kuruluş, güvenli kod incelemecisinin uyması gereken standartlar ve politikalar oluşturmalıdır. Bu, göreve gereken önemin verilmesini sağlayacak, böylece görev sadece işaretlenmesi gereken bir proje görevi olarak görülmeyecektir. Proje süresinin de göreve tahsis edilmesi gerekir, böylece görevlerin tamamlanması için (ve güvenli kod incelemesinden çıkan herhangi bir düzeltici görev için) yeterli zaman olur. Standartlar ayrıca yönetimin ve güvenlik uzmanlarının (örneğin CISO'lar, güvenlik mimarları) çalışanları hangi güvenli kodlamaya uyulması gerektiği konusunda yönlendirmesine ve çalışanların inceleme tahkimi gerektiğinde (standarda) başvurmasına olanak tanır.

**Kod İnceleme Raporları**

Standart bir rapor şablonu, kod gözden geçiren kişinin yazılım açıklarını uygulama tehdit modeline göre sınıflandırmasını ve önceliklendirmesini sağlamak için yeterli bilgi sağlayacaktır. Bu raporun sayfalar uzunluğunda olması gerekmez, belge tabanlı olabilir veya birçok otomatik kod inceleme aracına dahil edilebilir. Bir rapor aşağıdaki bilgileri sağlamalıdır:

- İnceleme tarihi.
- Uygulama adı, kod modülleri incelendi.
- Geliştiriciler ve kod gözden geçirenlerin isimleri.
- Görev veya özellik adı, (TFS, GIT, Subversion, sorun bileti, vb.).

- Varsa yazılım güvenlik açığını sınıflandırmak ve önceliklendirmek için kısa bir cümle(ler) ve varsa düzeltici görevlerin yerine getirilmesi veya takip edilmesi gerekenler.
- Gereksinimler, tasarım, test ve tehdit modelleme belgeleri dahil olmak üzere görev/özellik ile ilgili belgelere bağlantı.
- Kullanılıyorsa Kod İnceleme kontrol listesi veya kuruluş Kod İnceleme Kontrol Listesine bağlantı. (bkz. Ek A)
- Geliştiricinin kod üzerinde gerçekleştirdiği testler. Tercihen birim veya otomatik testlerin kendileri de inceleme sunumunun bir parçası olabilir.
- Kod incelemesinden önce FxCop, BinScope Binary Analyzer, vb. gibi herhangi bir araç kullanılmışsa.

Günümüzde çoğu kuruluş S-SDLC süreçlerine çevikliği eklemek için S-SDLC süreçlerini değiştirmiştir. Bu nedenle kuruluşun, güvenli kod incelemelerinin nerede ve ne sıklıkta yapılması gerektiğini en iyi şekilde belirlemek için kendi iç geliştirme uygulamalarına bakması gerekecektir. Eğer proje gecikmişse ve bütçe aşılmışsa, bu durum bir yazılım düzeltmesinin güvenli bir güvenlik açığına neden olma ihtimalini artırır çünkü artık projenin daha hızlı bir şekilde dağıtıma alınmasına odaklanılmaktadır. Üretimdeki kod için kod incelemeleri yazılım açıklarını bulabilir, ancak bilgisayar korsanları ile hatayı bulmak için bir yarış olduğunu ve düzeltici düzeltme üzerinde çalışılırken savunmasız yazılımın üretimde kalacağını anlayın.

### 6.3 Kod İncelemesi Ne Zaman Yapılmalı

Bir kuruluş kod incelemelerini dahili kod sürecinin bir parçası haline getirmeye karar verdiğinde. Sorulması gereken bir sonraki büyük soru, kodun SDLC'nin hangi aşamalarında gözden geçirileceğini belirlemektir. Bu bölümde kod incelemelerini dahil etmenin üç olası yolundan bahsedilmektedir. SDLC'de kodun gözden geçirilebileceği üç aşama vardır:

**Kod kontrol edilmek üzereyken (pre-commit)**

Geliştirme organizasyonu, kodun kaynak kodu deposuna gönderilmeden önce tüm kodun gözden geçirilmesi gerektiğini kendi süreçlerinde belirtebilir. Bu, gözden geçirme zaman alabileceği için check-in sürecini yavaşlatmak gibi bir dezavantaja sahiptir, ancak standardın altındaki kodun kod satırına asla yerleştirilmemesi ve yönetimin (süreçler takip ediliyorsa) gönderilen kodun öngörülen kalitede olduğundan emin olabilmesi gibi birçok avantajı vardır.

Örneğin, süreçler, sunulacak kodun gereksinimlere ve tasarım dokümantasyonuna ve gerekli birim ve otomatik testlere bağlantılar içermesi gerektiğini belirtebilir. Bu şekilde gözden geçirenler tam olarak yapılan kod değişikliği hakkında bilgi sahibi olacak (dokümantasyon sayesinde) ve geliştiricinin kodu nasıl test ettiğini bileceklerdir (testler sayesinde). Hakemler dokümantasyonun eksiksiz olduğunu ya da testlerin yeterince kapsamlı olduğunu düşünmezlerse, incelemeyi kodun kendisinden dolayı değil, gerekli dokümanlar ya da testler eksik olduğu için reddedebilirler. Otomatik testlerin gece boyunca çalıştığı CI kullanılan bir ortamda, geliştirme ekibi bir bütün olarak gönderilen kodun yeterli kalitede olup olmadığını ertesi gün (check-in'i takiben) bilecektir. Ayrıca yönetim, bir hata veya özellik kontrol edildiğinde geliştiricinin görevini tamamladığını bilir, geliştirme görevine risk ekleyen "Bu testleri gelecek hafta bitireceğim" senaryoları yoktur.

**Kod bir kod tabanına yeni kontrol edildiğinde (post-commit)**

Burada geliştirici kod değişikliğini gönderir ve ardından kod farkını incelemeye göndermek için kod deposu değişiklik listelerini kullanır. Bu yöntemin avantajı, geliştiricinin kodunu kontrol etmeden önce geçmesi gereken bir inceleme kapısı olmadığı için daha hızlı olmasıdır. Dezavantajı ise, uygulamada bu yöntemin daha düşük kalitede koda yol açabilmesidir. Bir geliştirme-

er, kod kontrol edildikten sonra daha küçük sorunları düzeltmeye daha az eğilimli olacaktır, genellikle "Kod artık içeride, iş görür." mantığıyla. Ayrıca zamanlama riski de vardır, çünkü diğer geliştiriciler inceleme tamamlanmadan veya değişiklikler ve testler yazılmadan önce aynı modüle başka kod düzeltmeleri yazabilir, bu da geliştiricinin yalnızca akran veya güvenlik incelemesinden gelen kod değişikliklerini uygulamak zorunda olmadığı, aynı zamanda bunu sonraki diğer değişiklikleri bozmayacak şekilde yapmak zorunda olduğu anlamına gelir. Birdenbire geliştirici, herhangi bir gerileme olmadığından emin olmak için sonraki düzeltmeleri yeniden test etmek zorunda kalır.

Çevik metodolojiyi kullanan bazı geliştirme kuruluşları süreçlerine bir 'güvenlik sprinti' ekler. Güvenlik sprinti sırasında kod güvenlik açısından gözden geçirilebilir ve güvenliğe özel test senaryoları (yazılı veya otomatik) eklenebilir.

**Kod denetimleri yapıldığında**
Bazı kuruluşların belirli aralıklarla (örneğin yıllık olarak) veya güvenlik açığı olan bir kod parçasının kod tabanında tekrarlandığından şüphelenildiğinde kodu gözden geçirme süreçleri vardır. Burada statik kod analizörleri veya kodda basit dize aramaları (belirli güvenlik açığı kalıpları için) süreci hızlandırabilir. Bu inceleme bir özelliğin veya hata düzeltmesinin sunulmasına bağlı değildir, süreçle ilgili hususlar tarafından tetiklenir ve muhtemelen tek bir gönderimin incelenmesinden ziyade tüm bir uygulamanın veya kod tabanının incelenmesini içerir.

### Güvenli Kod İncelemelerini Kimler Yapmalı?

Bazı kuruluşlar güvenli kod incelemesinin bir güvenlik veya risk analizi ekibi üyesinin işi olabileceğini varsaymaktadır. Ancak tüm geliştiricilerin uygulamalarının maruz kaldığı noktaları ve uygulamaları için hangi tehditlerin mevcut olduğunu anlamaları gerekir.

Birçok şirketin güvenlik ekiplerinde kodlama geçmişine sahip üyeler bulunmaz ve bu da geliştirme ekipleriyle etkileşimi zorlaştırabilir. Bu nedenle geliştirme ekipleri genellikle güvenlik girdilerine ve rehberliğine şüpheyle yaklaşır. Güvenlik ekipleri genellikle gizlilik ve bütünlük kontrollerinin yerinde olduğundan emin olmak için işleri yavaşlatmaya istekliyken, geliştiriciler destekledikleri iş birimlerinin mümkün olduğunca çabuk kod oluşturma ve güncelleme baskısıyla karşı karşıyadır. Ne yazık ki uygulama operasyonel veya iş ihtiyaçları açısından ne kadar kritikse, kodu üretime dağıtma baskısı da o kadar artar.

Güvenli kod incelemelerini SDLC süreçlerine dahil etmek en iyisidir, böylece geliştirme kuruluşları güvenliği bir engel olarak değil, bir yardımcı olarak görürler. Daha önce de belirtildiği gibi, güvenli kodlama KOBİ'lerini bir kuruluşun geneline yaymak (BSIMM terminolojisinde uydular), güvenli kod inceleme görevlerinin ölçeklenmesini ve daha fazla geliştirme ekibine ulaşmasını sağlar. Süreç büyüdükçe, daha fazla geliştirici güvenli kodlama sorunları hakkında farkındalık kazanacak (incelemeleri güvenli kodlama gerekçesiyle reddedildikçe) ve kod incelemelerindeki güvenli kodlama sorunlarının sıklığı azalacaktır.

## 6.4 Çevik ve Şelale Geliştirme için Güvenlik Kodu İncelemesi

Günümüzde çevik geliştirme, programlama, sürekli entegrasyon, test, proje yönetimi vb. gibi pek çok uygulama için kullanılan bir şemsiye terimdir. Çevik geliştirmenin pek çok çeşidi vardır, belki de uygulayıcı sayısı kadar çeşidi vardır. Çevik geliştirme, geliştirme ekibinin hangi uygulamaları kullanmak istediğini seçebildiği heterojen bir referans çerçevesidir.

Agile, kodun nasıl ve ne zaman gözden geçirileceğini etkileyebilecek bazı uygulamalara sahiptir; örneğin Agile, kod gözden geçirme ve test işlemlerini geliştirme aşamasına mümkün olduğunca yakın tutmaya çalışır. Kısa geliştirme döngüleri (diğer adıyla İterasyonlar veya Sprintler) tanımlamak yaygın bir uygulamadır. Her döngünün sonunda, tüm kod üretim kalitesinde olmalıdır. Eksik olabilir, ancak bir değer katmalıdır. Gözden geçirme sürekli olması gerektiğinden bu durum gözden geçirme sürecini etkiler. Güvenli kodlama incelemesi açısından bakıldığında, geliştirme sürecinin tamamlanmış olması bir fark yaratmamalıdır.

kuruluş çevik veya şelale geliştirme uygulamalarını kullanır. Kod incelemesi, özellik geliştirme ve test etme sırasına veya kodlama görevine atanan zaman modellerine göre değil, sunulan koda göre hizalanır. Birçok kuruluşta şelale ve çevik arasındaki çizgi bulanıklaşmakta, geleneksel şelale departmanları gece derlemeleri, otomatik testler, test güdümlü geliştirme vb. dahil olmak üzere çevik sürekli entegrasyon (CI) yönlerini tanıtmaktadır.

### 6.5 Kod İncelemesine Risk Tabanlı Bir Yaklaşım

Bir geliştirme şirketinde, yılda bir kez çalıştırılan arka uç komut dosyalarındaki basit tek satırlık hata düzeltmelerinden kritik iş mantığındaki büyük özellik gönderimlerine kadar çeşitli derecelerde kod değişiklikleri gözden geçirilir. Tipik olarak kod incelemesinin yoğunluğu, değişikliğin sunduğu algılanan riske bağlı olarak değişir.

Sonunda, kod incelemesinin ölçeği kaynakların (vasıflı kişiler, şirket zamanı, makineler, vb.) yönetimine bağlıdır. Bir üründe meydana gelen her kod değişikliği için birden fazla güvenlik uzmanı getirmek ölçeklenebilir olmayacaktır, bu kişilerin veya bu ekiplerin kaynakları her değişikliği ele alacak kadar büyük olmayacaktır. Bu nedenle şirketler hangi değişikliklerin önemli olduğuna ve yakından incelenmesi gerektiğine ve hangilerinin minimum denetimle geçmesine izin verilebileceğine karar verebilir. Bu, yönetimin geliştirme döngüsünü daha iyi boyutlandırmasını sağlayacaktır; eğer bir değişiklik yüksek riskli bir alanda yapılacaksa, yönetim kod incelemesi için yeterli zaman ayırmayı bilecek ve ilgili becerilere sahip kişilerin hazır bulunmasını sağlayacaktır. Hangi değişikliklerin hangi seviyede kod incelemesine ihtiyaç duyduğuna karar verme süreci, değişikliğin içinde bulunduğu modülün risk seviyesine dayanır.

Kod değişikliklerinin gözden geçirme yoğunluğu değiştirilen modülün risk düzeyine dayanıyorsa, risk düzeyine kim karar vermelidir? Nihayetinde yönetim bir şirketin çıktılarından sorumludur ve dolayısıyla şirket tarafından satılan ürünlerle ilgili riskten de sorumludur. Bu nedenle, bir kod değişikliği ile ilişkili riske karar vermek için tekrarlanabilir bir ölçü veya çerçeve oluşturmak yönetime (veya yönetim tarafından yetkilendirilen kişilere) bağlıdır.

Bir modülün veya kod parçasının riskine ilişkin kararlar sağlam bir fayda maliyet analizine dayanmalıdır ve tüm modüllerin yüksek riskli olduğuna karar vermek sorumsuzluk olacaktır. Bu nedenle yönetim, kod tabanını ve ürünlerin karşılaştığı güvenlik sorunlarını anlayan kişilerle bir araya gelmeli ve çeşitli kod unsurları için bir risk ölçüsü oluşturmalıdır. Kod modüllere, dizinlere, ürünlere vb. ayrılabilir ve her birinin risk seviyesi belirlenebilir.

Risk analizi alanında kuruluşlara risk atamak için çeşitli yöntemler mevcuttur ve bu tür tartışmalara birçok kitap ayrılmıştır. Riskin belirlenmesine yönelik üç ana teknik **aşağıdaki tablo 1'de** özetlenmiştir.

**Tablo 1:** Risk Belirleme Seçenekleri

| Teknik | Yöntem |
|---|---|
| Nicel | İnsanları bir araya getirin ve kodla ilişkili potansiyel kayıp için parasal bir değer belirleyin. Kodun tehlikeye girme olasılığını ölçün. Risk düzeyini belirlemek için bu hesaplamalardan elde edilen dolar değerlerini kullanın. |
| Niteliksel | İnsanları bir araya getirin ve modüllerle ilişkilendirilen kayıp düzeyine ve uzlaşma olasılığına ilişkin görüşleri tartışın. Niteliksel yaklaşım, kayıpla ilgili parasal ilişkileri belirlemeye çalışmaz, ancak ilişkili kayıpların algılanmasına veya görüşüne yönelir. |
| Delphi | Modüllerin kayıpları ve ödünleri konusunda insanlarla bağımsız olarak görüşün veya onları sorgulayın, bu arada geri bildirimin anonim olacağını bilmelerini sağlayın. Buradaki izlenim, insanların sorulara daha dürüst cevaplar vereceği ve diğer insanların argümanlarından ve cevaplarından etkilenmeyecekleri yönündedir. |

Risk, kötü bir şeyin gerçekleşme ihtimali ve gerçekleşmesi halinde yol açabileceği zarardır. Farklı kod modüllerinin risk profiline karar verme kriterleri, değişikliklerin gerçekleştirilmesinden sorumlu yönetim ekibine bağlı olacaktır, örnekler **tablo 2'de verilmiştir.**

**Tablo 2:** Bir Kod Modülünün Risk Profilinin Belirlenmesi İçin Ortak Kriterler

| Kriterler | Açıklama |
|---|---|
| Maruz kalma kolaylığı | Kod değişikliği doğrudan internete maruz kalan bir kod parçasında mı yapılıyor? İçeriden biri arayüzü doğrudan kullanıyor mu? |
| Kayıp değeri | Modüle bir güvenlik açığı eklenirse ne kadar kayıp yaşanabilir? Modül kritik bir parola karma mekanizması mı içeriyor yoksa bazı dahili test araçlarında HTML sınırında basit bir değişiklik mi? |
| Düzenleyici kontroller | Bir kod parçası, uyulması gereken bir standartla ilişkili iş mantığını uyguluyorsa, uygunsuzluğun cezaları yüksek olabileceğinden bu mod-üller yüksek riskli olarak kabul edilebilir. |

Risk seviyeleri ürünler ve modüllerle ilişkilendirildiğinde, hangi seviyede kod incelemesi yapılması gerektiğini belirleyen politikalar oluşturulabilir. Birinci seviye risk modülündeki kod değişikliklerinin bir Güvenlik Mimarı da dahil olmak üzere 3 kişi tarafından gözden geçirilmesi gerekirken, 4. seviye risk modülündeki değişikliklerin yalnızca bir kişilik hızlı bir akran incelemesine ihtiyacı olabilir.

Daha riskli modüller için diğer seçenekler (veya kriterler), otomatik test veya statik analiz taleplerini içerebilir; örneğin, yüksek riskli koddaki kod değişiklikleri, statik analiz araçlarında %80 kod kapsamı ve gerileme olmadığından emin olmak için yeterli otomatik eşleştirilmiş testler içermelidir. Bu kriterler, değiştirilen kodu test edebildiklerinden emin olmak için kod incelemesinin bir parçası olarak talep edilebilir ve kontrol edilebilir.

Bazı şirketler kodlarını mantıksal olarak farklı depolara bölerek daha hassas kodların sınırlı sayıda geliştiricinin erişebildiği bir depoda yer almasını sağlar. Kod bu şekilde bölünmüşse, yalnızca daha riskli koda erişimi olan geliştiricilerin bu kod üzerinde inceleme yapabilmesi gerektiği unutulmamalıdır.

Risk analizi, **tablo 3'te olduğu** gibi, ürüne risk getiren bir kod değişikliğine nasıl tepki verileceğine karar vermek için kod incelemesi sırasında da kullanılabilir. Tipik bir risk analizi sürecinde, ekibin riskleri kabul etme, transfer etme, kaçınma veya azaltma konusunda karar vermesi gerekir. Kod incelemeleri söz konusu olduğunda riski transfer etmek mümkün değildir çünkü riski transfer etmek normalde maruz kalınan maliyeti karşılamak için sigorta yaptırmak anlamına gelir.

**Tablo 3:** Kod İncelemesinde Belirlenen Risklerin Ele Alınması İçin

| Risk Çözümü | Açıklama |
|---|---|
| Azaltma | Bu tipik çözüm yoludur. Bir kod gözden geçiricisi, kod değişikliğinin iş mantığının bir unsuruna (veya sadece bir hataya) risk getirdiğini tespit ettiğinde, hatayı veya kodu riski azaltacak şekilde düzeltmek için kod değiştirilecektir. |
| Kabul et | Kod değişikliği kodda bir risk oluşturuyorsa ancak iş mantığını uygulamanın başka bir yolu yoksa, risk kabul edilebilir olarak değerlendirilirse kod değişikliği kod incelemesinden geçebilir. Göz ardı edilmemesi için risk ve her türlü geçici çözüm veya hafifletici faktör doğru şekilde belgelenmelidir. |
| Kaçının | Kod değişikliği kabul edilemeyecek kadar büyük bir risk getirdiğinde ve bir kod değişikliği uygulayarak riski azaltmak mümkün olmadığında, ekibin değişikliği gerçekleştirmemeyi düşünmesi gerekir. İdeal olarak bu karara kod gözden geçirme aşamasından önce varılmalıdır, ancak kod uygulaması sırasında bir kod modülünün anlaşılan risk profilini değiştiren ve yönetimi bir değişikliğin yapılıp yapılmaması gerektiğini yeniden düşünmeye sevk eden faktörlerin ortaya çıkması tamamen mümkündür. |

### 6.6 Kod İnceleme Hazırlığı

Uygulamanın güvenlik incelemesi, yaygın güvenlik hatalarının yanı sıra uygulamanın iş mantığına özgü sorunları da ortaya çıkarmalıdır. Bir kod bütününü etkili bir şekilde gözden geçirmek için gözden geçirenlerin uygulamanın iş amacını ve kritik iş etkilerini anlaması önemlidir. Gözden geçirenler saldırı yüzeyini anlamalı, farklı tehdit unsurlarını ve bunların motivasyonlarını ve uygulamaya potansiyel olarak nasıl saldırabileceklerini belirlemelidir.

Kodu incelenen yazılım geliştiricisi için kod incelemesi yapmak bir denetim gibi gelebilir ve geliştiriciler geri bildirimi kişisel olarak almamakta zorlanabilir. Buna yaklaşmanın bir yolu, gözden geçiren, geliştirme ekibi, iş temsilcileri ve diğer ilgili taraflar arasında bir işbirliği ortamı yaratmaktır. Bir polis değil de bir danışman imajı çizmek, geliştirme ekibinden işbirliği almak için önemlidir.

Bilgi toplamanın ne ölçüde gerçekleşeceği kurumun büyüklüğüne, yeniden gözden geçirenlerin beceri setine ve gözden geçirilen kodun kritikliğine/riskine bağlı olacaktır. CSS dosyasında 20 kişilik bir başlangıçta yapılacak küçük bir değişiklik, tam bir tehdit modeline ve ayrı bir güvenli inceleme ekibine yol açmayacaktır. Aynı zamanda milyarlarca dolarlık bir şirkette yeni bir tek oturum açma kimlik doğrulama modülü, bir zamanlar güvenli kodlama üzerine bir makale okumuş bir kişi tarafından incelenen güvenli kod olmayacaktır. Aynı kuruluş içinde bile, yüksek riskli modüller veya uygulamalar tehdit modeline tabi tutulabilirken, daha düşük riskli modüller, gözden geçirenin modülün güvenlik modelini anlamasına daha az vurgu yapılarak gözden geçirilebilir.

Bu bölüm, gözden geçirenin (veya gözden geçirme ekibinin) güvenli kod gözden geçirmesine tabi tutulan uygulama hakkında anlamaya çalışması gereken temel öğeleri sunacaktır. Bu, tam bir güvenlik taban çizgisi oluşturacak kaynaklara sahip olmayan küçük şirketlerde veya daha büyük şirketlerdeki düşük riskli kodlarda kullanılabilir. Daha sonraki bir bölümde, daha büyük şirketler tarafından en yüksek riskli kod tabanlarında kullanılacak olan tehdit modellemesi ayrıntılı olarak ele alınacaktır.

İdeal bir dünyada, gözden geçiren kişi uygulamanın tasarım aşamasına dahil olur, ancak bu durum nadiren gerçekleşir. Ancak kod değişikliğinin boyutu ne olursa olsun, kod incelemesini başlatan mühendis, incelemecileri ilgili mimari veya tasarım belgelerine yönlendirmelidir. Bunu yapmanın en kolay yolu, ilk e-postaya veya kod inceleme aracına belgelere (çevrimiçi bir belge havuzunda depolandıkları takdirde) bir bağlantı eklemektir. Gözden geçiren kişi daha sonra temel risklerin güvenlik kontrolleri tarafından uygun şekilde ele alındığını ve bu kontrollerin doğru yerlerde kullanıldığını doğrulayabilir.

İncelemeyi etkin bir şekilde yürütmek için incelemeyi yapan kişi aşağıdaki hususlara aşinalık geliştirmelidir:

**Uygulama özellikleri ve İş Kuralları**

Gözden geçiren kişi, uygulama tarafından halihazırda sağlanan tüm özellikleri anlamalı ve bunlarla ilgili tüm iş kısıtlamalarını/kurallarını yakalamalıdır. Ayrıca, bir uygulamanın yol haritasında olabilecek gelecekteki potansiyel işlevsellik konusunda dikkatli olmak ve böylece mevcut kod incelemeleri sırasında alınan güvenlik kararlarını geleceğe dönük hale getirmek için bir durum vardır. Bu sistemin başarısız olmasının sonuçları nelerdir? Uygulama işlevlerini amaçlandığı gibi yerine getiremezse kurum bundan büyük ölçüde etkilenir mi?

**Bağlam**

Tüm güvenlik, güvence altına almaya çalıştığımız şey bağlamındadır. Elma satan bir uygulama için askeri standart güvenlik mekanizmaları önermek aşırıya kaçmak ve bağlamdan kopmak olur. Ne tür veriler manipüle ediliyor veya işleniyor ve bu veriler tehlikeye girerse şirkete ne gibi zararlar verir? Bağlam, güvenli kod denetimi ve risk değerlendirmesinin "Kutsal Kase "sidir.

**Hassas Veriler**
Gözden geçiren kişi, uygulama için hassas olan hesap numaraları ve şifreler gibi veri varlıklarını da not etmelidir. Veri varlıklarının hassasiyetlerine göre kategorize edilmesi, gözden geçirenin uygulamadaki herhangi bir veri kaybının etkisini belirlemesine yardımcı olacaktır.

**Kullanıcı rolleri ve erişim hakları**
Uygulamaya erişmesine izin verilen kullanıcı türünü anlamak önemlidir. Uygulama dışarıya mı yoksa "güvenilir" kullanıcılara mı açık? Genel olarak, yalnızca bir kuruluşun dahili kullanıcılarının erişebildiği bir uygulama, internetteki herkesin erişebildiği bir uygulamadan farklı tehditlere maruz kalabilir. Bu nedenle, uygulamanın kullanıcılarını ve konuşlandırıldığı ortamı bilmek, gözden geçirenin tehdit unsurlarını doğru bir şekilde fark etmesini sağlayacaktır. Buna ek olarak, uygulamada mevcut olan farklı ayrıcalık seviyeleri de anlaşılmalıdır. Bu, incelemeyi yapan kişinin uygulama için geçerli olabilecek farklı güvenlik ihlallerini/ayrıcalık yükseltme saldırılarını sıralamasına yardımcı olacaktır.

**Uygulama türü**
Bu, uygulamanın tarayıcı tabanlı bir uygulama mı, masaüstü tabanlı bağımsız bir uygulama mı, bir web hizmeti mi, mobil uygulamalar mı yoksa hibrit bir uygulama mı olduğunu anlamak anlamına gelir. Farklı uygulama türleri farklı güvenlik tehditleriyle karşı karşıyadır ve uygulamanın türünü anlamak, gözden geçirenin belirli güvenlik kusurlarını aramasına, doğru tehdit unsurlarını belirlemesine ve uygulamaya uygun gerekli kontrolleri vurgulamasına yardımcı olacaktır.

**Kod**
Kullanılan dil(ler), bu dilin güvenlik perspektifinden özellikleri ve sorunları. Bir programcının dikkat etmesi gereken konular ve güvenlik ve performans açısından dilin en iyi uygulamaları.

**Tasarım**
MVC tasarım prensibi kullanılarak geliştirilen web uygulamaları genellikle iyi tanımlanmış bir kod düzenine sahiptir. Uygulamalar kendi özel tasarımlarına sahip olabilir veya Struts/Spring vb. gibi bazı iyi bilinen tasarım çerçevelerini kullanabilirler. Uygulama özellikleri/yapılandırma parametreleri nerede saklanır? Herhangi bir özellik/URL için iş sınıfı nasıl tanımlanır? Herhangi bir isteği işlemek için ne tür sınıflar çalıştırılır? (örn. merkezi denetleyici, komut sınıfları, görünüm sayfaları vb.) Herhangi bir istek için görünüm kullanıcılara nasıl gösterilir?

**Şirket Standartları ve Yönergeleri**
Birçok şirkette yönetim tarafından dikte edilen standartlar ve kılavuzlar bulunur. Bu, yönetimin (kuruluşun bilgi güvenliğinden büyük ölçüde sorumlu olan) çeşitli işlevlere hangi güvenlik seviyelerinin uygulanacağını ve bunların nasıl uygulanması gerektiğini kontrol etme şeklidir. Örneğin, şirketin Güvenli Kodlama Yönergeleri belgesi varsa, gözden geçirenler yönergeleri bilmeli, anlamalı ve kod gözden geçirme sırasında bunları uygulamalıdır.

### 6.7 Kod İncelemesi Keşif ve Bilgi Toplama

Gözden geçirenlerin etkili olabilmeleri için uygulama hakkında belirli bilgilere ihtiyaçları olacaktır. Bu bilgiler sıklıkla tasarım belgeleri, iş gereksinimleri, işlevsel özellikler, test sonuçları ve benzerleri incelenerek elde edilebilir. Ancak gerçek dünyadaki projelerin çoğunda dokümantasyon önemli ölçüde güncel değildir ve neredeyse hiçbir zaman uygun güvenlik bilgilerine sahip değildir. Geliştirme organizasyonunun mimari ve tasarım belgeleri için prosedürleri ve şablonları varsa, gözden geçiren kişi bu aşamalarda güvenliğin dikkate alınmasını (ve belgelenmesini) sağlamak için güncellemeler önerebilir.

Gözden geçirenler başlangıçta uygulamaya aşina değillerse, başlamanın en etkili yollarından biri geliştiricilerle ve uygulamanın baş mimarıyla konuşmaktır. Bunun uzun bir toplantı olması gerekmez, geliştirme ekibinin temel güvenlik hususları hakkında bazı temel bilgileri paylaşması için bir beyaz tahta oturumu olabilir

ve kontroller. Uygulamanın nasıl çalıştığına dair gözden geçirenlere iyi bir fikir vermek için gerçekte çalışan uygulamanın gözden geçirilmesi çok faydalıdır. Ayrıca, kod tabanının yapısına ve kullanılan kütüphanelere kısa bir genel bakış da gözden geçirenlerin başlamasına yardımcı olabilir.

Uygulama hakkında bilgi başka bir şekilde elde edilemiyorsa, gözden geçirenlerin keşif yapmak ve kodu inceleyerek uygulamanın nasıl çalıştığına dair bilgi paylaşmak için biraz zaman harcaması gerekecektir. Tercihen bu bilgiler daha sonra gelecekteki incelemelere yardımcı olmak üzere belgelenebilir.

Güvenlik kodu incelemesi sadece kod yapısıyla ilgili değildir. Verileri hatırlamak önemlidir; kodu gözden geçirmemizin nedeni, para, fikri mülkiyet, ticari sırlar veya yaşamlar gibi emanet edilen bilgileri ve varlıkları yeterince koruduğundan emin olmaktır. Uygulamanın işlemesi amaçlanan verilerin bağlamı, potansiyel riskin belirlenmesinde çok önemlidir. Eğer uygulama yerleşik/bilinen bir tasarım çerçevesi kullanılarak geliştirilmişse, bu soruların çoğunun cevabı önceden tanımlanmış olacaktır. Ancak, özel olması durumunda, bu bilgi, özellikle veri akışını ve dahili doğrulamaları yakalamada inceleme sürecine kesinlikle yardımcı olacaktır. Uygulamanın mimarisini bilmek, uygulama için geçerli olabilecek güvenlik tehditlerini anlamada uzun bir yol kat eder.

Tasarım, bir uygulamanın taslağıdır; geliştirilmesi için bir temel oluşturur. Uygulamanın düzenini gösterir ve bunun için gereken farklı uygulama bileşenlerini tanımlar. Uygulamanın yürütme akışını belirleyen bir yapıdır. Uygulama tasarımlarının çoğu MVC kavramına dayanmaktadır. Bu tür tasarımlarda farklı bileşenler, herhangi bir kullanıcı talebine hizmet etmek için sıralı bir şekilde birbirleriyle etkileşime girer. Tasarım incelemesi güvenli yazılım geliştirme sürecinin ayrılmaz bir parçası olmalıdır. Tasarım incelemeleri güvenlik gereksinimlerinin daha iyi bir şekilde uygulanmasına da yardımcı olur.

Önerilen tasarımın amacını anlamak için akış şemaları, sıra diyagramları, sınıf diyagramları ve gereksinim belgeleri dahil olmak üzere önerilen tasarımın tüm gerekli bilgilerinin toplanması. Tasarım, özellikle veri akışı, farklı uygulama bileşenleri etkileşimleri ve veri işleme açısından kapsamlı bir şekilde incelenir. Bu, manuel analiz ve tasarım veya teknik mimarın ekibiyle yapılan tartışmalar yoluyla gerçekleştirilir. Uygulamada güvenlik ihlallerine yol açabilecek hassas alanları analiz etmek için uygulamanın tasarımı ve mimarisi iyice anlaşılmalıdır.

Tasarımı anladıktan sonra, bir sonraki aşama tasarıma yönelik tehditleri analiz etmektir. Bu, tasarımın bir saldırganın bakış açısından gözlemlenmesini ve içinde bulunan arka kapıların ve güvensiz alanların ortaya çıkarılmasını içerir. **Tablo 4** aşağıda

**Tablo 4:** Güvenli Kod İncelemesi Sırasında Örnek Tasarım Soruları

| Tasarım Alanı | Dikkate alınması gereken sorular |
|---|---|
| Veri Akışı | • Kullanıcı girdileri doğrudan iş mantığına referans vermek için mi kullanılıyor?<br>• Veri bağlama kusurları için potansiyel var mı?<br>• Arıza durumlarında yürütme akışı doğru mu? |
| Kimlik doğrulama ve erişim kontrolü | • Tasarım tüm kaynaklar için erişim kontrolü uyguluyor mu?<br>• Oturumlar doğru şekilde işleniyor mu?<br>• Kimlik doğrulama olmadan hangi işlevlere erişilebilir? |
| Mevcut güvenlik kontrolleri | • Üçüncü taraf güvenlik kontrollerinde bilinen herhangi bir zayıflık var mı<br>• Güvenlik kontrollerinin yerleşimleri doğru mu? |
| Mimarlık | • Harici sunuculara bağlantılar güvenli mi?<br>• Dış kaynaklardan gelen girdiler doğrulanıyor mu? |
| Yapılandırma dosyaları ve veri depoları | • Yapılandırma dosyalarında hassas veriler var mı?<br>• Yapılandırma veya veri dosyalarına kimin erişimi var? |

**Kod İnceleme Kontrol Listesi**

Geliştirme ekibinin doldurabileceği genel bir kontrol listesi tanımlamak, gözden geçirenlere istenen bağlamı sağlayabilir. Kontrol listesi, geliştiricilerin denedikleri ya da düşündükleri güvenlik seviyesi için iyi bir barometredir. Güvenlik kodu incelemesi yaygın bir gereklilik haline gelirse, bu kontrol listesi bir geliştirme prosedürüne (örneğin belge şablonları) dahil edilebilir, böylece bilgiler kod incelemecileri tarafından her zaman kullanılabilir. Örnek bir kod inceleme kontrol listesi için Ek A'ya bakınız.

Kontrol listesi, aşağıdaki gibi en kritik güvenlik kontrollerini ve güvenlik açığı alanlarını kapsamalıdır:

- Veri Doğrulama
- Kimlik Doğrulama
- Oturum Yönetimi
- Yetkilendirme
- Kriptografi
- Hata İşleme
- Günlük kaydı
- Güvenlik Yapılandırması
- Ağ Mimarisi

güvenli kod incelemelerine yardımcı olmak için mimariye ve tasarıma sorulabilecek bazı soruları vurgular.
Her güvenlik gereksinimi, tasarım için en uygun güvenlik kontrolüyle ilişkilendirilmelidir. Burada, herhangi bir gereksinimi karşılamak veya bir tehdidi azaltmak için tasarıma dahil edilmesi gereken kesin değişiklikleri veya eklemeleri belirleriz. Güvenlik gereksinimleri listesi ve önerilen kontroller daha sonra geliştirme ekipleriyle tartışılabilir. Ekiplerin soruları ele alınmalı ve kontrollerin dahil edilmesinin fizibilitesi belirlenmelidir. Varsa istisnalar dikkate alınmalı ve alternatif öneriler sunulmalıdır. Bu aşamada güvenlik kontrolleri üzerinde nihai bir anlaşma sağlanır. Geliştirme ekipleri tarafından dahil edilen nihai tasarım tekrar gözden geçirilebilir ve daha sonraki geliştirme süreci için son haline getirilebilir.

### 6.8 Statik Kod Analizi

Statik Kod Analizi, S-SDLC'nin uygulama aşamasında gerçekleştirilir. Statik kod analizi genellikle 'statik' (çalışmayan) kaynak kod içindeki olası güvenlik açıklarını vurgulamaya çalışan statik kod analiz araçlarının çalıştırılması anlamına gelir.

İdeal olarak statik kod analizi araçları güvenlik açıklarını otomatik olarak ve çok az sayıda yanlış pozitifle bulur. Bu, bulduğu hataların gerçek hatalar olduğuna dair yüksek derecede güvene sahip olması gerektiği anlamına gelir. Ancak bu ideal, birçok uygulama güvenlik açığı türü için mevcut teknolojinin ötesindedir. Bu nedenle, bu tür araçlar genellikle bir analistin tüm kusurları otomatik olarak bulan bir araçtan ziyade kusurları daha verimli bir şekilde bulabilmeleri için kodun güvenlikle ilgili bölümlerini sıfırlamalarına yardımcı olan yardımcılar olarak hizmet eder.

Uygulamada güvensiz kod, tasarım veya yapılandırma nedeniyle hatalar olabilir. Aşağıdaki iki seçenekten birini kullanarak hataları tespit etmek için uygulama kodu üzerinde otomatik analiz gerçekleştirilebilir:

### 1. Desen aramasına dayalı statik kod tarayıcı komut dosyaları (kurum içi ve açık kaynak).

### 2. Statik kod analizörleri (ticari ve açık kaynak).

Kaynak kodu tarayıcılarının avantaj ve dezavantajları **tablo 5 ve 6**'da gösterilmektedir.

**Tablo 5:** Kaynak Kod Tarayıcıları Kullanmanın Avantajları

| Avantaj | Açıklama |
|---|---|
| Manuel çabalarda azalma | Taranacak kalıpların türü uygulamalar arasında ortak olmaya devam etmektedir, bilgisayarlar bu tür taramalarda insanlardan daha iyidir. Bu senaryoda tarayıcılar, büyük kod tabanlarında güvenlik açıklarını arama sürecini otomatikleştirerek büyük bir rol oynamaktadır. |
| Güvenlik açıklarının tüm örneklerini bulun | Tarayıcılar, belirli bir güvenlik açığının tüm örneklerini tam konumlarıyla birlikte tanımlamada çok etkilidir. Bu, tüm dosyalardaki kusurları izlemenin zor olduğu daha büyük kod tabanı için yararlıdır. |
| Kaynaktan lavaboya analiz | Bazı analizörler kodu izleyebilir ve kaynaktan lavaboya analiz yoluyla güvenlik açıklarını belirleyebilir. Uygulamaya olası girdileri belirler ve bunları herhangi bir güvensiz kod modeliyle ilişkili bulana kadar kod boyunca ayrıntılı bir şekilde izlerler. Böyle bir kaynaktan lavaboya analiz, geliştiricilerin kusurun tam bir kök neden analizini elde ettikleri için kusurları daha iyi anlamalarına yardımcı olur |
| Ayrıntılı raporlama formatı | Tarayıcılar, gözlemlenen güvenlik açıkları hakkında tam kod parçacıkları, risk derecelendirmesi ve güvenlik açıklarının tam açıklamasını içeren ayrıntılı bir rapor sunar. Bu, geliştirme ekiplerinin kusurları kolayca anlamasına ve gerekli kontrolleri uygulamasına yardımcı olur |

Kod tarama komut dosyaları ve açık kaynak araçları güvensiz kod kalıplarını bulmada etkili olsalar da, genellikle veri akışını izleme yeteneğinden yoksundurlar. Bu boşluk, kodu kısmen (veya tamamen) derleyerek ve yürütme dallarını inceleyerek güvensiz kod kalıplarını belirleyen ve kaynaktan batışa analize izin veren statik kod analizörleri tarafından doldurulur. Statik kod analizörleri ve tarayıcılar, kod inceleme sürecini tamamlayan kapsamlı seçeneklerdir.

**Tablo 6:** Kaynak Kod Tarayıcıları Kullanmanın

| Sınırlama | Açıklama |
|---|---|
| İş mantığı kusurları dokunulmadan kalır | Uygulamanın iş mantığı, işlemleri ve hassas verileriyle ilgili kusurlar tarayıcılar tarafından dokunulmadan kalır. Uygulamanın özelliklerine ve tasarımına özgü olarak uygulanması gereken güvenlik kontrolleri genellikle tarayıcılar tarafından işaret edilmemektedir. Bu durum statik kod analizörlerinin en büyük kısıtlaması olarak değerlendirilmektedir. |
| Sınırlı kapsam | Statik kod analizörleri genellikle belirli çerçeveler veya diller için tasarlanmıştır ve bu kapsam dahilinde belirli bir dizi güvenlik açığı kalıbını arayabilirler. Bu kapsamın dışında, arama kalıbı havuzlarında yer almayan sorunları ele almakta başarısız olurlar. |
| Tasarım kusurları | Tasarım kusurları kod yapısına özgü değildir ve statik kod analizörleri koda odaklanır. Bir tarayıcı/analizör koda bakarken bir tasarım sorununu tespit edemezken, bir insan genellikle uygulamalarına bakarken tasarım sorunlarını tespit edebilir. |
| Yanlış pozitifler | Statik kod analizörleri tarafından işaretlenen tüm sorunlar gerçekten sorun değildir ve bu nedenle bu araçlardan elde edilen sonuçların güvenli kodlamayı anlayan deneyimli bir programcı tarafından anlaşılması ve önceliklendirilmesi gerekir. Bu nedenle, güvenli kod kontrolünün otomatikleştirilebileceğini ve derlemenin sonunda çalıştırılabileceğini uman herkes hayal kırıklığına uğrayacaktır ve analizörlerle hala bir miktar manuel müdahale gerekmektedir. |

### Statik analiz aracı seçme

Çok fazla seçenek olduğu için statik analiz aracı seçmek zor bir iştir. Aşağıdaki karşılaştırma tabloları, bu liste kapsamlı olmasa da, kuruluşun kendileri için hangi aracın doğru olduğuna karar vermesine yardımcı olabilir.

**Bir araç seçmek için bazı kriterler şunlardır:**
• Araç kullanılan programlama dilini destekliyor mu?

• Ticari veya ücretsiz araçlar arasında bir tercih var mı? Genellikle ticari araçlar ücretsiz olanlardan daha fazla özelliğe sahiptir ve daha güvenilirdir, ancak kullanılabilirlikleri farklı olabilir.

• Ne tür bir analiz gerçekleştiriliyor? Güvenlik, kalite, statik veya dinamik analiz mi?

Bir sonraki adım, oldukça öznel olduğu için bazı çalışmaların yapılmasını gerektirir. Yapılacak en iyi şey, ekibin kullanıcı deneyimi, güvenlik açıklarının raporlanması, yanlış pozitiflerin seviyesi ve özelleştirme ve müşteri desteği gibi farklı yönlerden memnun olup olmadığını görmek için birkaç aracı test etmektir. Seçim, özelliklerin sayısına göre değil, ihtiyaç duyulan özelliklere ve bunların S-SDLC'ye nasıl entegre edilebileceğine göre yapılmalıdır. Ayrıca, aracı seçmeden önce, uygun bir araç seçmek için hedeflenen kullanıcıların uzmanlığı açıkça değerlendirilmelidir.

### 6.9 Uygulama Tehdit Modellemesi

Tehdit modellemesi, bir uygulamanın güvenliğini analiz etmek için derinlemesine bir yaklaşımdır. Çalışanların bir uygulamayla ilişkili güvenlik risklerini tanımlamalarını, ölçmelerini ve ele almalarını sağlayan yapılandırılmış bir yaklaşımdır. Tehdit modellemesi, kodu gözden geçirmeye yönelik bir yaklaşım değildir, ancak uygulamanın bağlamını ve risk analizini sağlayarak güvenli kod gözden geçirme sürecini tamamlar.

Tehdit modellemesinin S-SDLC'ye dahil edilmesi, uygulamaların en başından itibaren güvenlik ön planda tutularak geliştirilmesini sağlamaya yardımcı olabilir. Bu, tehdit modelleme sürecinin bir parçası olarak üretilen dokümantasyonla birlikte, gözden geçirenin sistemi daha iyi anlamasını sağlayabilir, gözden geçirenin uygulamaya giriş noktalarının nerede olduğunu (yani saldırı yüzeyi) ve her giriş noktasıyla ilişkili tehditleri (yani saldırı vektörleri) görmesini sağlar.

Tehdit modellemesi kavramı yeni değildir, ancak son yıllarda açık bir zihniyet değişikliği olmuştur. Modern tehdit modellemesi, bir sisteme savunmacının bakış açısının aksine potansiyel bir saldırganın bakış açısından bakar. Aralarında tehdit modellemesini S-SDLC'lerinin temel bir bileşeni haline getiren Microsoft'un da bulunduğu pek çok şirket son yıllarda bu sürecin güçlü savunucuları olmuştur ve son yıllarda ürünlerinin artan güvenliğinin nedenlerinden biri olduğunu iddia etmektedirler.

Mevcut uygulamalar gibi S-SDLC dışında kaynak kodu analizi yapıldığında, tehdit modellemesinin sonuçları risk temelli bir yaklaşımı teşvik ederek kaynak kodu analizinin karmaşıklığını azaltmaya yardımcı olur. Gözden geçiren kişi, tüm kaynak kodunu eşit odaklanma ile gözden geçirmek yerine, tehdit modellemesinin yüksek riskli tehditlerle sıraladığı bileşenlerin güvenlik kodu gözden geçirmesine öncelik verebilir.

**Tehdit modelleme süreci 3 üst düzey adıma ayrıştırılabilir:**

#### 6.9.1. Adım 1: Uygulamayı Ayrıştırın.

Tehdit modelleme sürecindeki ilk adım, uygulamanın ve dış varlıklarla nasıl etkileşime girdiğinin anlaşılmasıyla ilgilidir. Bu, uygulamanın nasıl kullanıldığını anlamak için kullanım durumları oluşturmayı, potansiyel bir saldırganın uygulama ile nerede etkileşime girebileceğini görmek için giriş noktalarını tanımlamayı, varlıkları, yani saldırganın ilgileneceği öğeleri / alanları tanımlamayı ve uygulamanın harici varlıklara vereceği erişim haklarını temsil eden güven düzeylerini belirlemeyi içerir. Bu bilgiler tehdit modeli belgesinde belgelenir ve aynı zamanda uygulama için veri akış diyagramları (DFD'ler) üretmek için kullanılır. DFD'ler sistemdeki farklı veri yollarını göstererek ayrıcalık (güven) sınırlarını vurgular.

**Uygulamayı ayrıştırırken göz önünde bulundurulması gereken öğeler arasında Harici Bağımlılıklar**

Dış bağımlılıklar, uygulama için tehdit oluşturabilecek, uygulamanın kodu dışındaki öğelerdir. Bu öğeler genellikle hala kuruluşun kontrolü altındadır, ancak muhtemelen geliştirme ekibinin kontrolü altında değildir. Dış bağımlılıkları araştırırken bakılması gereken ilk alan, uygulamanın üretim ortamında nasıl konuşlandırılacağıdır.

Bu, uygulamanın nasıl çalıştırılmak istendiğine veya istenmediğine bakmayı içerir. Örneğin, uygulamanın kuruluşun sertleştirme standardına göre sertleştirilmiş bir sunucuda çalıştırılması bekleniyorsa ve bir güvenlik duvarının arkasında yer alması bekleniyorsa, bu bilgi belgelenmelidir.

**Giriş Noktaları**

Giriş noktaları (diğer adıyla saldırı vektörleri), potansiyel saldırganların uygulama ile etkileşime girebileceği veya ona veri sağlayabileceği arayüzleri tanımlar. Potansiyel bir saldırganın bir uygulamaya saldırabilmesi için giriş noktalarının mevcut olması gerekir. Bir uygulamadaki giriş noktaları katmanlı olabilir, örneğin bir web uygulamasındaki her web sayfası birden fazla giriş noktası içerebilir.

**Varlıklar**

Sistem, saldırganın ilgilendiği bir şeye sahip olmalıdır; bu öğeler/ilgi alanları varlık olarak tanımlanır. Varlıklar esasen tehdit hedefleridir, yani tehditlerin var olma nedenidirler. Varlıklar hem fiziksel varlıklar hem de soyut varlıklar olabilir. Örneğin, bir uygulamanın varlığı müşterilerin listesi ve kişisel bilgileri olabilir; bu fiziksel bir varlıktır. Soyut bir varlık ise bir kurumun itibarı olabilir.

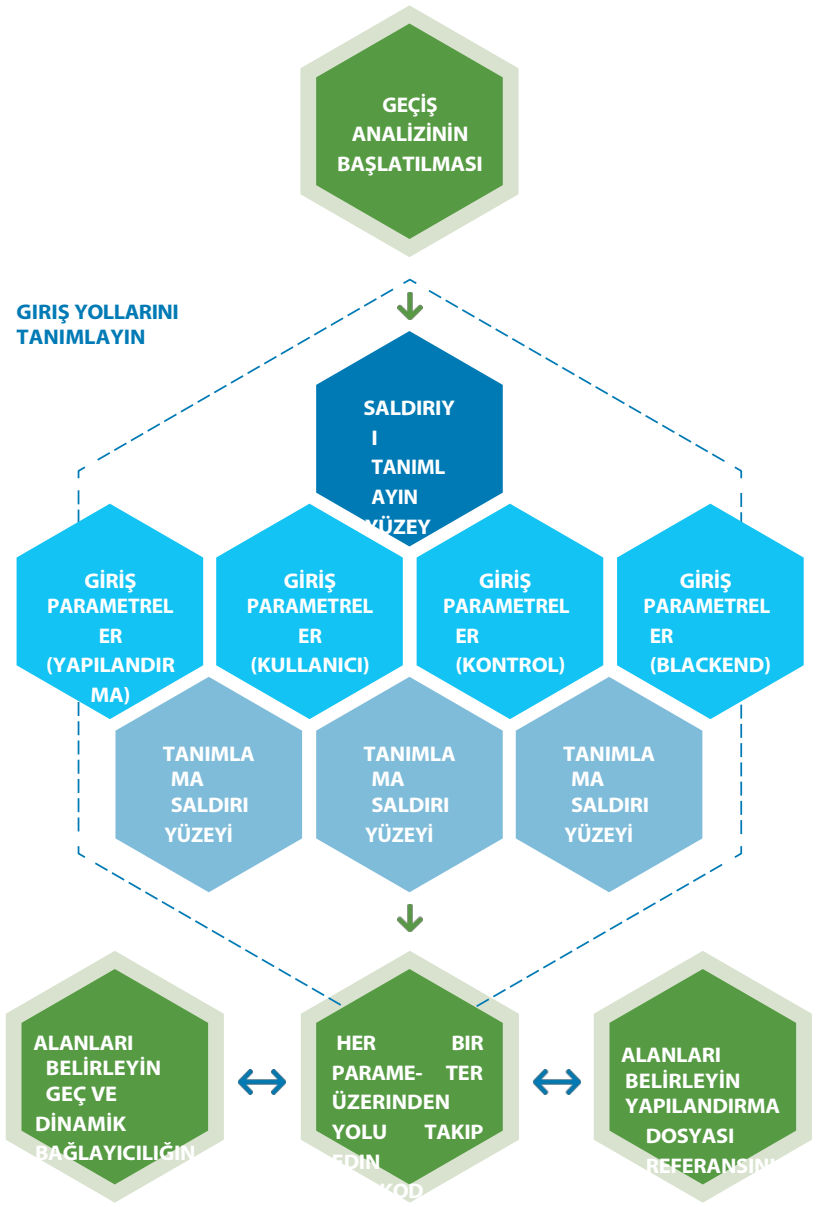
**Saldırı Yüzeyinin Belirlenmesi**

Saldırı yüzeyi, girdiler, veri akışları ve işlemler analiz edilerek belirlenir. Bir güvenlik kodu incelemesi gerçekleştirmenin önemli bir kısmı saldırı yüzeyinin analizini yapmaktır. Bir uygulama girdileri alır ve bir tür çıktı üretir. İlk adım, koda giden tüm girdileri tanımlamaktır.

Uygulamanın girdileri aşağıdaki madde işaretlerini içerebilir ve **şekil 4'te** bir uygulamanın girdi yollarını tanımlamaya yönelik örnek bir süreç açıklanmaktadır:

- Tarayıcı girişi
- Çerezler
- Mülkiyet dosyaları
- Dış süreçler
- Veri beslemeleri
- Hizmet yanıtları
- Düz dosyalar
- Komut satırı parametreleri
- Ortam değişkenleri

**Şekil 4:** Girdi yollarının belirlenmesine yönelik örnek süreç



**Güven Seviyeleri**
Güven seviyeleri, uygulamanın harici varlıklara vereceği erişim haklarını temsil eder. Güven seviyeleri, giriş noktaları ve varlıklarla çapraz referanslıdır. Bu, bir ekibin her giriş noktasında gereken erişim haklarını veya ayrıcalıklarını ve her varlıkla etkileşim için gerekenleri tanımlamasına olanak tanır.

**Veri akışı analizi**
Saldırı yüzeyinin araştırılması dinamik ve statik veri akışı analizini içerir. Değişkenlerin nerede ve ne zaman ayarlandığı ve değişkenlerin iş akışı boyunca nasıl kullanıldığı, nesnelerin ve parametrelerin niteliklerinin program içindeki diğer verileri nasıl etkileyebileceği. Parametrelerin, yöntem çağrılarının ve veri alışverişi mekanizmalarının gerekli güvenliği uygulayıp uygulamadığını belirler.

**İşlem analizi**
İşlem analizi, çağrılan ilgili güvenlik işlevleriyle birlikte uygulama içindeki tüm işlemleri tanımlamak ve analiz etmek için gereklidir.

İşlem analizi sırasında ele alınan alanlar şunlardır:

- Veri/Girdi Tüm güvenilmeyen kaynaklardan gelen verilerin doğrulanması
- Kimlik Doğrulama
- Oturum Yönetimi
- Yetkilendirme
- Kriptografi (duran ve aktarılan veriler)
- Hata İşleme / Bilgi Sızıntısı
- Günlüğe Kaydetme / Denetleme

**Veri Akış Diyagramları**
Toplanan tüm bilgiler, Veri Akış Diyagramları (DFD'ler) kullanılarak uygulamanın doğru bir şekilde modellenmesini sağlar. DFD'ler, uygulamanın verileri nasıl işlediğine dair görsel bir temsil sunarak çalışanın uygulamayı daha iyi anlamasını sağlayacaktır. DFD'lerin odak noktası, verilerin uygulama içinde nasıl hareket ettiği ve hareket ederken verilere ne olduğudur. DFD'ler hiyerarşik yapıdadır, bu nedenle uygulamayı alt sistemlere ayırmak için kullanılabilirler. Üst düzey DFD, çalışanın modellenmekte olan uygulamanın kapsamını netleştirmesini sağlayacaktır. Daha düşük seviyeli yinelemeler, belirli verilerin işlenmesinde yer alan belirli süreçlere daha fazla odaklanılmasını sağlayacaktır.

Aşağıdaki **tablo 7'**de gösterildiği gibi, DFD'lerde tehdit modellemesi için kullanılan bir dizi sembol vardır:

**Tablo 7:** Tehdit Modelleme Sembolleri

| ELEMENT | GÖRÜNTÜ | AÇIKLAMA |
|---|---|---|
| HARICI VARLIK |  | Harici varlık şekli, bir giriş noktası aracılığıyla uygulamayla etkileşime giren uygulama dışındaki herhangi bir varlığı temsil etmek için kullanılır. |
| SÜREÇ |  | İşlem şekli, uygulama içinde verileri işleyen bir görevi temsil eder. Görev, verileri işleyebilir veya verilere dayalı bir eylem gerçekleştirebilir. |

| | | |
|---|---|---|
| **ÇOKLU SÜREÇ** |  | Çoklu süreç şekli, bir alt süreçler koleksiyonunu sunmak için kullanılır. Çoklu süreç başka bir DFD'de alt süreçlerine ayrılabilir. |
| **VERİ DEPOSU** |  | Veri deposu şekli, verilerin depolandığı konumları temsil etmek için kullanılır. Veri depoları verileri değiştirmez, yalnızca verileri depolar. |
| **VERİ AKIŞI** |  | Veri akışı şekli, uygulama içindeki veri hareketini temsil eder. Veri hareketinin yönü ok ile g ö s t e r i l i r . |
| **AYRICALIK SINIRI** |  | Ayrıcalık sınırı şekli, veriler uygulama boyunca akarken ayrıcalık seviyelerinin değişimini temsil etmek için kullanılır. |

DFD'ler verilerin sistem içinde mantıksal olarak nasıl hareket ettiğini gösterir ve verilerin depolanması ve bu bileşenler aracılığıyla kontrol akışıyla birlikte sisteme giren veya sistemden çıkan verilerin tanımlanmasını sağlar. Güven sınırları, güven seviyesinin değiştiği tüm konumları gösterir. İşlem bileşenleri web sunucuları, uygulama sunucuları ve veritabanı sunucuları gibi verilerin nerede işlendiğini gösterir. Giriş noktaları verilerin sisteme girdiği yerleri (örn. giriş alanları, yöntemler), çıkış noktaları ise sistemden çıktığı yerleri (örn. dinamik çıktılar, yöntemler) gösterir. Giriş ve çıkış noktaları bir güven sınırı tanımlar.

### 6.9.2 Adım 2: Tehditleri belirleyin ve sıralayın

Tehditlerin tanımlanmasında kritik olan, bir tehdit kategorizasyon metodolojisinin kullanılmasıdır. STRIDE gibi bir tehdit kategorizasyonu veya Au- diting & Logging, Authentication, Authorization, Configuration Management, Data Protection in Storage and Transit, Data Validation ve Exception Management gibi tehdit kategorilerini tanımlayan Application Security Frame (ASF) kullanılabilir.

Tehdit kategorizasyonunun amacı, hem saldırgan (STRIDE) hem de savunma perspektifinden (ASF) tehditlerin belirlenmesine yardımcı olmaktır. Adım 1'de üretilen DFD'ler, veri kaynakları, süreçler, veri akışları ve kullanıcılarla etkileşimler gibi potansiyel tehdit hedeflerinin saldırganın bakış açısından belirlenmesine yardımcı olur. Bu tehditler ayrıca tehdit ağaçlarının kökleri olarak tanımlanabilir; her tehdit hedefi için bir ağaç vardır.

Savunma perspektifinden bakıldığında, ASF kategorizasyonu tehditlerin, bu tehditlere yönelik güvenlik kontrollerinin zayıflıkları olarak tanımlanmasına yardımcı olur. Örnekler içeren ortak tehdit listeleri bu tür tehditlerin tanımlanmasına yardımcı olabilir. Kullanım ve kötüye kullanım vakaları, mevcut koruyucu önlemlerin nasıl atlanabileceğini veya bu tür bir korumanın nerede eksik olduğunu gösterebilir.

Her bir tehdit için güvenlik riskinin belirlenmesi, DREAD gibi değere dayalı bir risk modeli veya genel risk faktörlerine (örneğin olasılık ve etki) dayalı daha az öznel nitel bir risk modeli kullanılarak belirlenebilir.

Tehditlerin belirlenmesindeki ilk adım, bir tehdit kategorizasyonunun benimsenmesidir. Bir tehdit kategorizasyonu pro-

tehditlerin uygulamada yapılandırılmış ve tekrarlanabilir bir şekilde sistematik olarak tanımlanabilmesi için ilgili örneklerle birlikte bir dizi tehdit kategorisi sunar.

**STRIDE**

STRIDE modeline dayalı tehdit listeleri, saldırganın hedeflerine ilişkin tehditlerin tanımlanmasında faydalıdır. Örneğin, tehdit senaryosu oturum açmaya saldırmaksa, saldırgan kimlik doğrulamasını kırmak için parolayı kaba kuvvetle zorlar mı? Tehdit senaryosu, başka bir kullanıcının ayrıcalıklarını elde etmek için ayrıcalıkları yükseltmeye çalışmaksa, saldırgan zorla tarama yapmaya çalışır mı?

STRIDE gibi bir tehdit kategorizasyonu, saldırgan hedeflerini **tablo 8'de** gösterildiği gibi sınıflandırarak tehditlerin tanımlanmasında faydalıdır.

**Tablo 8:** Adım Özelliklerinin Açıklanması

| STRIDE | Açıklama |
|---|---|
| Spoofing | "Kimlik sahteciliği" çok sayıda kullanıcısı olan ancak uygulama ve veritabanı düzeyinde tek bir yürütme bağlamı sağlayan uygulamalar için önemli bir risktir. Özellikle, kullanıcılar başka bir kullanıcıya dönüşememeli veya başka bir kullanıcının niteliklerini üstlenememelidir. |
| Kurcalama | Kullanıcılar kendilerine teslim edilen verileri potansiyel olarak değiştirebilir, iade edebilir ve böylece istemci tarafı doğrulama, GET ve POST sonuçları, çerezler, HTTP başlıkları ve benzerlerini potansiyel olarak manipüle edebilir. Uygulama ayrıca kullanıcıdan alınan verileri dikkatli bir şekilde kontrol etmeli ve saklamadan ya da kullanmadan önce akla uygun ve uygulanabilir olduğunu doğrulamalıdır. |
| Reddetme | Kullanıcılar, faaliyetlerine ilişkin yeterli denetim veya kayıt tutulmaması durumunda işlemlere itiraz edebilir. Örneğin, bir kullanıcı finansal bir transfer yapmadığını söylerse ve işlevsellik uygulama aracılığıyla faaliyetlerini takip edemezse, işlemin zarar olarak yazılması son derece muhtemeldir. |
| Bilgi Açıklaması | Kullanıcılar bir sisteme özel bilgilerini gönderme konusunda haklı olarak temkinlidir. Bir saldırganın anonim olarak veya yetkili bir kullanıcı olarak kullanıcı verilerini herkese açık bir şekilde ifşa etmesi mümkün müdür? |
| Hizmet Reddi | Uygulama tasarımcıları, uygulamalarının bir hizmet reddi saldırısına maruz kalabileceğinin farkında olmalıdır. Büyük dosyalar, karmaşık hesaplamalar, ağır aramalar veya uzun sorgular gibi pahalı kaynakların kullanımı, kimliği doğrulanmış ve yetkili kullanıcılar için ayrılmalı ve anonim kullanıcılar tarafından kullanılmamalıdır. |
| Ayrıcalığın Yükseltilmesi | Bir uygulama farklı kullanıcı ve yönetici rolleri sağlıyorsa, kullanıcının rolünü daha yüksek bir ayrıcalığa yükseltemeyeceğinden emin olmak çok önemlidir. |

Tüm olası saldırı vektörlerinin saldırganın bakış açısından değerlendirilmesi hayati önem taşımaktadır. Örneğin, oturum açma sayfası kimlik doğrulama bilgilerinin gönderilmesine izin verir ve bir giriş noktası tarafından kabul edilen giriş verileri SQL enjeksiyonu, siteler arası komut dosyası oluşturma ve arabellek taşmaları gibi güvenlik açıklarından yararlanmak için potansiyel kötü amaçlı girdiler için doğrulanmalıdır. Ek olarak, bu noktadan geçen veri akışı, akış boyunca sonraki bileşenlere giriş noktalarına yönelik tehditleri belirlemek için kullanılmalıdır. Eğer sonraki bileşenler kritik olarak kabul edilebiliyorsa (örneğin hassas verileri tutuyorsa), bu giriş noktası da daha kritik olarak kabul edilebilir. Uçtan uca bir veri akışında, doğrulama yapılmadan aktarılan bir oturum açma sayfasındaki giriş verileri (örn. kullanıcı adı ve parola), kimlik doğrulamasını kırmak veya veritabanındaki bir tabloyu değiştirmek için bir sorgu oluşturmak üzere bir SQL enjeksiyon saldırısı için kullanılabilir.

Çıkış noktaları istemciye yönelik saldırı noktaları (örn. XSS güvenlik açıkları) olarak hizmet edebileceği gibi bilgi ifşası güvenlik açıklarının gerçekleşmesine de neden olabilir. Gizli verileri işleyen bileşenlerden (örn. veri erişim bileşenleri) çıkış noktaları söz konusu olduğunda, gizliliği ve bütünlüğü korumak için güvenlik kontrollerinden yoksun olan çıkış noktaları, bu tür gizli bilgilerin yetkisiz bir kullanıcıya ifşa edilmesine yol açabilir.

Birçok durumda çıkış noktaları tarafından etkinleştirilen tehditler ilgili giriş noktasının tehditleriyle ilişkilidir. Oturum açma örneğinde, çıkış noktası aracılığıyla kullanıcıya döndürülen hata mesajları, giriş noktası saldırılarına izin verebilir,

örneğin hesap

toplama (örn. kullanıcı adı bulunamadı) veya SQL enjeksiyonu (örn. SQL istisna hataları). Savunma perspektifinden bakıldığında, ASF gibi güvenlik kontrol kategorizasyonları tarafından yönlendirilen tehditlerin tanımlanması, bir tehdit analistinin güvenlik kontrollerindeki zayıflıklarla (örneğin güvenlik açıkları) ilgili belirli konulara odaklanmasını sağlar. Tipik olarak tehdit tanımlama süreci, başlangıçta her bir bileşen için geçerli olan tehdit listesindeki tüm olası tehditlerin değerlendirildiği yinelemeli döngülerden geçmeyi içerir. Bir sonraki yinelemede tehditler, saldırı yolları, tehdidin istismar edilmesine yol açan temel nedenler (örn. güvenlik açıkları) ve gerekli hafifletme kontrolleri (örn. karşı önlemler) araştırılarak daha fazla analiz edilir.

Yaygın tehditler, güvenlik açıkları ve saldırılar değerlendirildikten sonra, daha odaklı bir tehdit analizi kullanım ve kötüye kullanım durumlarını dikkate almalıdır. Kullanım senaryolarının derinlemesine analiz edilmesiyle, bir tehdidin gerçekleşmesine yol açabilecek zayıflıklar tespit edilebilir. Kötüye kullanım durumları, güvenlik gereksinimi mühendisliği faaliyetinin bir parçası olarak belirlenmelidir. Bu kötüye kullanım vakaları, mevcut koruyucu önlemlerin nasıl atlanabileceğini veya bu tür bir korumanın nerede eksik olduğunu gösterebilir. Son olarak, ayrıştırılmış sistemin her bir bileşenine yönelik tehdit türlerini belirleyerek tüm bunları bir araya getirmek mümkündür. Bu, STRIDE veya ASF gibi bir tehdit kategorizasyonu, tehdidin güvenlik açığı ile nasıl açığa çıkabileceğini belirlemek için tehdit ağaçlarının kullanımı ve tehdidi azaltacak bir karşı önlemin eksikliğini daha da doğrulamak için kullanım ve kötüye kullanım durumları kullanılarak daha düşük seviyeli bir tehdit modelinde daha önce tartışılan teknikler tekrarlanarak yapılabilir.

**Microsoft DREAD tehdit-risk sıralama modeli**

Microsoft DREAD tehdit-risk sıralama modelinde, etki için teknik risk faktörleri Hasar ve Etkilenen Kullanıcılar iken, istismar kolaylığı faktörleri Yeniden Üretilebilirlik, İstismar Edilebilirlik ve Keşfedilebilirliktir. Bu risk faktörizasyonu, bir tehdidin farklı etki faktörlerine değer atanmasını sağlar.

Bir tehdidin sıralamasını belirlemek için, tehdit analistinin her bir risk faktörü için temel soruları cevaplaması gerekir, örneğin

**Tablo 9:** Dehşet Niteliklerinin Açıklanması

| DREAD | Sorular |
|---|---|
| Hasar | Saldırı başarılı olsaydı hasar ne kadar büyük olurdu?<br>Bir saldırgan sistemi tamamen ele geçirebilir ve manipüle edebilir mi? Bir saldırgan sistemi çökertebilir mi? |
| Tekrar Üretilebilirlik | Bir saldırıyı işe yarayacak şekilde yeniden üretmek ne kadar kolay?<br>Bu istismar otomatikleştirilebilir mi? |
| İstismar Edilebilirlik | Tehditten yararlanmak için ne kadar zaman, çaba ve uzmanlık gerekiyor?<br>Saldırganın kimliğinin doğrulanması gerekiyor mu? |
| Etkilenen Kullanıcılar | Bir tehdit istismar edilirse, kullanıcıların yüzde kaçı etkilenir?<br>Bir saldırgan sisteme yönetici erişimi sağlayabilir mi? |
| Keşfedilebilirlik | Bir saldırganın bu tehdidi keşfetmesi ne kadar kolay? |

Etki esas olarak hasar potansiyeline ve bir tehditten etkilenen bileşen sayısı gibi etkinin boyutuna bağlıdır.

Bu sorular, Olasılık ve Etki faktörlerine Yüksek, Orta ve Düşük gibi nitel değerler atayarak genel risk değerlerinin hesaplanmasına yardımcı olur. Bu durumda, DREAD modelinde olduğu gibi sayısal değerler yerine nitel değerlerin kullanılması, sıralamanın aşırı öznel olmasını önlemeye yardımcı olur.

**Olasılık**

Daha genel bir risk modeli Olasılığı (örneğin saldırı olasılığı) ve Etkiyi (örneğin hasar potansiyeli) dikkate alır:

**Risk = Olasılık x Etki**

Bunun kavramsal bir formül olduğunu ve olasılık ve etki için gerçek değerlerin kullanılmasının beklenmediğini unutmayın. Olasılık veya ihtimal, esas olarak tehdit türüne ve sistem özelliklerine bağlı olan istismar kolaylığı ve uygun bir karşı önlemin varlığıyla belirlenen bir tehdidi gerçekleştirme olasılığı ile tanımlanır.

### 6.9.3 Adım 3: Karşı önlemleri ve hafifletici tedbirleri belirleyin.

Bir tehdide karşı koruma eksikliği, bir karşı önlemin uygulanmasıyla risk maruziyeti azaltılabilecek bir güvenlik açığına işaret edebilir. Bu tür karşı önlemler tehdit-karşı önlem eşleme listeleri kullanılarak belirlenebilir. Tehditlere bir risk sıralaması atandıktan sonra, tehditleri en yüksek riskten en düşük riske doğru sıralamak ve belirlenen karşı önlemleri uygulayarak bu tür tehditlere yanıt vermek gibi hafifletme çabalarına öncelik vermek mümkündür.

Risk azaltma stratejisi, bu tehditleri oluşturdukları iş etkisi açısından değerlendirmeyi ve riski azaltmak için karşı önlemler (veya tasarım değişiklikleri) oluşturmayı içerebilir.

Diğer seçenekler arasında riskin kabul edilmesi, iş etkisinin kontroller nedeniyle kabul edilebilir olduğunun varsayılması, kullanıcının tehdit hakkında bilgilendirilmesi, tehdidin yarattığı riskin tamamen ortadan kaldırılması veya en az tercih edilen seçenek olan hiçbir şey yapmamak yer alabilir. Belirlenen risk aşırı ise, bir şeylerin yanlış gitme riski faydadan daha büyük olduğu için işlevsellik veya ürün durdurulabilir.

Karşı önlem tanımlamasının amacı, daha önce tehdit analizi yoluyla belirlenen her bir tehdidin gerçekleşmesini önleyebilecek bir tür koruyucu önlemin (örn. güvenlik kontrolü, politika önlemleri) mevcut olup olmadığını belirlemektir. Bu durumda zafiyetler, karşı önlemi olmayan tehditlerdir.

Bu tehditlerin her biri STRIDE veya ASF ile kategorize edildiğinden, uygulamada verilen kategori dahilinde uygun karşı önlemler bulmak mümkün olabilir. Yukarıdaki adımların her biri gerçekleştirildikçe belgelenir. Sonuçta ortaya çıkan belge seti, uygulama için tehdit modelidir. Tehdit modellemesinin nasıl yapılacağına dair ayrıntılı örnekler **Ek B'**de verilmiştir.

**Tehdit Profili**

Tehditler ve ilgili karşı önlemler belirlendikten sonra, aşağıdaki kriterlerle bir tehdit profili türetmek mümkündür:

**Tablo 10:** Azaltılmış Tehdit Türleri

| Tehdit Türü | Açıklama |
|---|---|
| Hafifletilmemiş tehditler | Karşı önlemi olmayan ve tamamen istismar edilebilecek ve bir etkiye neden olabilecek güvenlik açıklarını temsil eden tehditler. |
| Kısmen hafifletilmiş tehditler | Bir veya daha fazla karşı önlemle kısmen hafifletilen tehditler, yalnızca kısmen istismar edilebilen ve sınırlı bir etkiye neden olan güvenlik açıklarını temsil eder. |
| Tamamen azaltılmış tehditler | Bu tehditlerin uygun karşı önlemleri vardır ve güvenlik açığını ortaya çıkarmaz ve etkiye neden olmaz. |

## 6.10 Metrikler ve Kod İncelemesi

Metrikler bir kod parçasının boyutunu ve karmaşıklığını ölçer. Kodu gözden geçirirken dikkate alınabilecek kalite ve güvenlik özelliklerinin uzun bir listesi vardır (doğruluk, verimlilik, taşınabilirlik, sürdürülebilirlik, güvenilirlik ve güvenlik gibi ancak bunlarla sınırlı olmayan). Hiçbir iki kod inceleme oturumu aynı olmayacaktır, bu nedenle en iyi yola karar vermek için biraz muhakeme gerekecektir. Metrikler kod incelemesinin ölçeğine karar verilmesine yardımcı olabilir.

Kod gözden geçirenlerin performansı ve gözden geçirme sürecinin doğruluğu, kod gözden geçirme işlevinin performansı ve kod gözden geçirme işlevinin verimliliği ve etkinliği ile ilgili metrikler de kaydedilebilir.

**Şekil 5**, kod inceleme süreci boyunca metriklerin kullanımını açıklamaktadır.

**Bir inceleme görevinin boyutunu hesaplamaya yönelik seçeneklerden bazıları şunlardır:**

**Kod Satırları (LOC):**
Çalıştırılabilir kod satırlarının sayımı (yorumlanmış kod veya boş satırlar sayılmaz). Bu kabaca bir tahmin verir ancak özellikle bilimsel değildir.

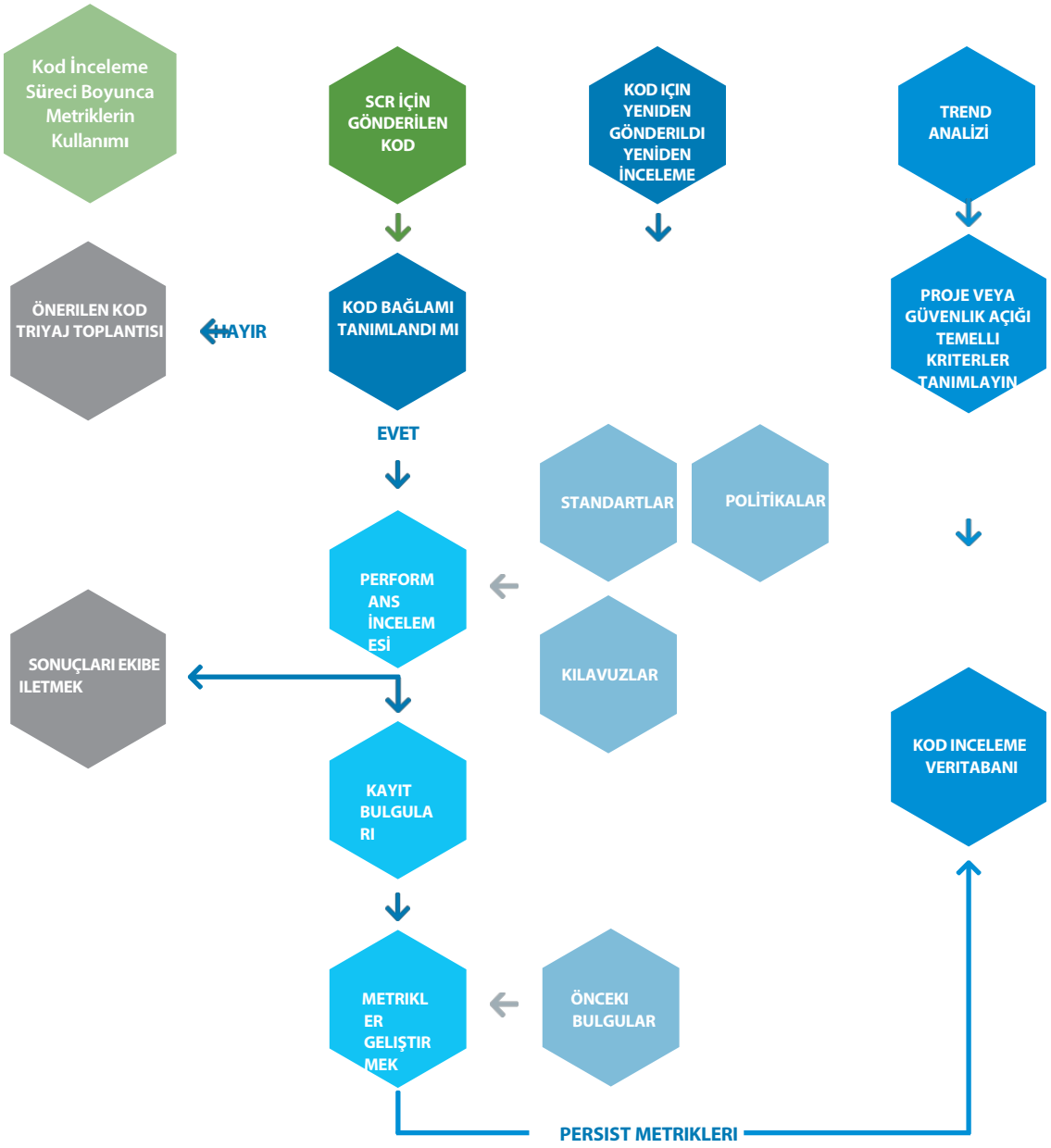
**İşlev Noktası:**
İşlevselliği ölçerek yazılım boyutunun tahmin edilmesi. Kullanılan programlama dilinden veya geliştirme metodolojisinden bağımsız olarak belirli bir görevi yerine getiren bir dizi ifadenin kombinasyonu. Nesne yönelimli bir dilde bir sınıf işlevsel bir nokta olabilir.

**Kusur Yoğunluğu:**
Kod Satırı (LOC) başına ortalama programlama hatası oluşumu. Bu, kod kalitesinin üst düzey bir görünümünü verir ancak daha fazlasını vermez. Hata yoğunluğu tek başına pragmatik bir metriğe yol açmaz. Hata yoğunluğu, koddaki büyük güvenlik kusurlarının yanı sıra küçük sorunları da kapsayacaktır; hepsi aynı şekilde ele alınır. Kodun güvenliği tek başına hata yoğunluğu kullanılarak doğru bir şekilde değerlendirilemez.

**Risk Yoğunluğu:**
Hata yoğunluğuna benzer, ancak keşfedilen sorunlar riske göre derecelendirilir (yüksek, orta ve düşük). Bunu yaparken, dahili uygulama geliştirme politikaları ve standartları tarafından tanımlanan [X Risk / LoC] veya [Y Risk / Fonksiyon Noktası] değeri (X&Y yüksek, orta veya düşük risklerdir) aracılığıyla geliştirilmekte olan kodun kalitesi hakkında fikir verebiliriz.

**Örneğin:**
**1000 (Kod Satırı) başına 4 Yüksek Riskli Hata**
**3 İşlev Noktası başına 2 Orta Riskli Hata**

**Şekil 5:** Girdi yollarının belirlenmesine yönelik örnek süreç



**Döngüsel karmaşıklık (CC):**
Bir sınıf, yöntem veya hatta tam bir sistem gibi bir kod öğesi üzerinde risk ve kararlılık tahminlerinin oluşturulmasına yardımcı olmak için kullanılan bir statik analiz metriğidir. Thomas McCabe tarafından 70'li yıllarda tanımlanmıştır ve hesaplanması ve uygulanması kolaydır, dolayısıyla kullanışlıdır.

McCabe döngüsel karmaşıklık metriği, bir programın test edilebilirliğini, anlaşılabilirliğini ve sürdürülebilirliğini göstermek için tasarlanmıştır. Bu, anlama, test etme, bakım vb. zorluklarını tahmin etmek için kontrol akış yapısının ölçülmesiyle gerçekleştirilir. Kontrol akış yapısı bir kez anlaşıldığında, programın ne ölçüde hata içerme olasılığı olduğu anlaşılabilir. Döngüsel karmaşıklık metriği, bir program modülü boyunca doğrusal olarak bağımsız yolların sayısını ölçen dil ve dil formatından bağımsız olarak tasarlanmıştır. Bu aynı zamanda test edilmesi gereken minimum yol sayısıdır.

Ürünün döngüsel karmaşıklığı bilinerek, en yüksek karmaşıklığa sahip modüle odaklanılabilir. Bu, büyük olasılıkla verilerin izleyeceği yollardan biri olacak ve böylece kişiyi güvenlik açıkları için potansiyel olarak yüksek riskli bir konuma yönlendirebilecektir. Karmaşıklık arttıkça daha fazla hata olma potansiyeli de artar. Daha fazla hata daha fazla güvenlik açığı olasılığını artırır.

Siklomatik karmaşıklık güvenlik riskini ortaya çıkarır mı? Modülün güvenlik duruşu gözden geçirilene kadar bunu bilemeyiz. Döngüsel karmaşıklık metriği, kodu incelemeye ve analiz etmeye nereden başlanacağı konusunda risk temelli bir yaklaşım sağlar. Bir uygulamanın güvenliğini sağlamak karmaşık bir iştir ve yazılım karmaşıklığı yazılım hatalarının tespit edilmesini zorlaştırabileceğinden karmaşıklık birçok yönden güvenliğin düşmanıdır. Yazılımın karmaşıklığı, ürün güncellendikçe veya bakımı yapıldıkça zaman içinde artar.

**Döngüsel karmaşıklık şu şekilde hesaplanabilir:**
**CC = Karar sayısı +1**
... bir kararın, yürütmenin if/else, switch, case, catch, while, do, şablonlu sınıf çağrıları vb. ile dallara ayrıldığı komutlar olarak kabul edileceği yer,

Karar sayısı arttıkça, karmaşıklık ve yol sayısı da artar. Karmaşık kod daha az kararlılığa ve sürdürülebilirliğe yol açar.

**Kod ne kadar karmaşıksa, hata riski de o kadar yüksektir. Bir şirket, bir modül için döngüsel karmaşıklık eşikleri belirleyebilir:**
**0-10: Kararlı kod, kabul edilebilir**
**karmaşıklık 11-15: Orta Risk, daha**
**karmaşık**
**16-20: Yüksek Riskli kod, bir kod birimi için çok fazla karar.**

Çok yüksek döngüsel karmaşıklığa sahip modüller son derece karmaşıktır ve daha küçük yöntemlere dönüştürülebilir.

**Kötü Düzeltme Olasılığı:**
Bu, bazı şirketlerde regresyon olarak bilinen, önceki bir hatayı düzeltmeye çalışırken yanlışlıkla bir programa eklenen bir hata olasılığıdır.
**Döngüsel Karmaşıklık: 1 - 10 == Kötü Düzeltme Olasılığı: %5**
**Döngüsel Karmaşıklık: 20 -30 == Kötü Düzeltme Olasılığı: %20**
**Döngüsel Karmaşıklık: > 50 == Kötü Düzeltme Olasılığı: %40**
**Döngüsel Karmaşıklık: 100'e Yaklaşıyor == Kötü Düzeltme Olasılığı: %60**

Yazılımın karmaşıklığı arttıkça yeni hataların ortaya çıkma olasılığı da artar.

**Muayene Oranı:**
Bu metrik, bir kod incelemesi gerçekleştirmek için gereken süre hakkında kabaca bir fikir edinmek için kullanılabilir. İnceleme oranı, bir kod inceleme uzmanının birim zamanda kapsayabileceği kapsam oranıdır. Örneğin, saat başına 250 satırlık bir oran temel alınabilir. Bu oran, inceleme kalitesinin bir ölçüsü olarak değil, sadece görevin süresini belirlemek için kullanılmalıdır.

**Kusur Tespit Oranı:**
Bu metrik birim zamanda bulunan hataları ölçer. Yine, kod inceleme ekibinin performansını ölçmek için kullanılabilir, ancak bir kalite ölçütü olarak kullanılmamalıdır. Hata tespit oranı normalde inceleme oranı (yukarıda) azaldıkça artacaktır.

**Yeniden İnceleme Kusur Oranı:**
Kodun yeniden incelenmesi sonucunda daha fazla kusurun bulunması, bazı kusurların hala mevcut olması veya daha önce keşfedilen kusurların (regresyonlar) giderilmeye çalışılmasıyla başka kusurların ortaya çıkma oranı.

### 6.11 Tarama Kodu

Kod tarama, inceleme hedefinin kod tabanını ve arayüz giriş noktalarını tarayarak olası güvenlik açığının bulunabileceği kilit kod işaretçilerini arama uygulamasıdır. Belirli API'ler dış dünya ile etkileşim veya dosya IO veya kullanıcı yönetimi ile ilgilidir ve bunlar bir saldırganın odaklanması gereken kilit alanlardır. Kod tararken bu alanlarla ilgili API'leri ararız. Güvenlik sorunlarına neden olabilecek iş mantığı alanlarını da aramamız gerekir, ancak bunlar genellikle ısmarlama adlara sahip ısmarlama yöntemlerdir ve belirli bir anahtar API ile ilişkileri nedeniyle belirli yöntemlere dokunabilsek bile doğrudan korunamazlar.

Ayrıca, belirli bir dille ilgili yaygın sorunları da aramamız gerekir; güvenlikle ilgili olmayabilecek, ancak olağanüstü durumlarda uygulamanın kararlılığını / kullanılabilirliğini etkileyebilecek sorunlar. Bir kod incelemesi gerçekleştirirken karşılaşılan diğer sorunlar, kişinin fikri mülkiyetini korumak için basit bir telif hakkı bildirimi gibi alanlardır. Genel olarak bu konular şirketlerin Kodlama Kılavuzlarının (veya Standartlarının) bir parçası olmalı ve kod incelemesi sırasında uygulanabilir olmalıdır. Örneğin, bir gözden geçiren, kodun mevcut haliyle çalışıp çalışmayacağına bakmaksızın, Kodlama Yönergelerindeki bir şeyi ihlal ettiği için bir kod incelemesini reddedebilir.

Kod taraması manuel olarak veya otomatik araçlar kullanılarak otomatik bir şekilde yapılabilir. Ancak manuel olarak çalışmak muhtemelen etkili değildir, çünkü (aşağıda görülebileceği gibi) bir dile uygulanabilecek çok sayıda gösterge vardır. Grep veya wingrep gibi basit araçlar kullanılabilir. Belirli bir programlama diliyle ilgili anahtar kelimeleri arayan başka araçlar da mevcuttur. Bir ekip, incelemede vurgulanacak dizeleri belirlemesine izin veren belirli bir inceleme aracı kullanıyorsa (örneğin, pygments sözdizimi yüksek çakmak kullanan Python tabanlı inceleme araçları veya ekibin kaynak kodunu değiştirebileceği şirket içi bir araç), aşağıdaki listelerden ilgili dize göstergelerini ekleyebilir ve bunların gözden geçirenlere otomatik olarak vurgulanmasını sağlayabilir.

Kod incelemesinin temeli, uygulama güvenliğini etkileyebilecek kod alanlarını bulmak ve analiz etmektir. Kod gözden geçiren kişinin kodu, ne yapmak istediğini ve hangi bağlamda kullanılacağını tam olarak anladığını varsayarsak, öncelikle kod tabanını ilgilenilen alanlar için taraması gerekir.

**Ek C,** aşağıdaki programlama dillerinde kod taramasının nasıl yapılacağına dair pratik örnekler vermektedir:
- .Net
- Java
- ASP
- C++/Apache


A1

# A1 ENJEKSİYON

## 7.1 Genel Bakış

**Enjeksiyon Nedir?**

Enjeksiyon saldırıları, kötü niyetli bir kullanıcının bir uygulamanın davranışını değiştirmek için uygulamaya içerik ve komutlar eklemesine veya enjekte etmesine olanak tanır. Bu tür saldırılar yaygındır, bir bilgisayar korsanının bir web sitesinin savunmasız olup olmadığını test etmesi kolaydır ve saldırganın bundan faydalanması kolay ve hızlıdır. Günümüzde güncellenmemiş eski uygulamalarda çok yaygındırlar.

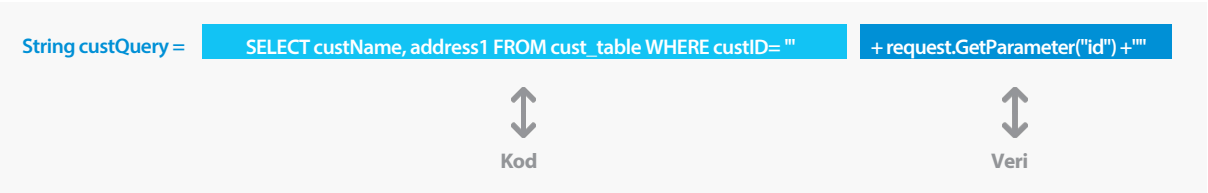
## 7.2 SQL Enjeksiyonu

En yaygın enjeksiyon zafiyeti SQL enjeksiyonudur. Enjeksiyon zafiyetinin düzeltilmesi ve buna karşı korunması da kolaydır. Bu güvenlik açığı SQL, LDAP, Xpath, OS komutları, XML ayrıştırıcıları kapsar.

**Enjeksiyon güvenlik açığı...**

**1.** Hassas bilgilerin ifşa edilmesi/sızdırılması.
**2.** Veri bütünlüğü sorunları. SQL enjeksiyonu verileri değiştirebilir, yeni veri ekleyebilir veya verileri silebilir.
**3.** Ayrıcalıkların yükseltilmesi.
**4.** Arka uç ağına erişim elde etme.

SQL komutları güvenilmeyen girdilerden korunmaz. SQL ayrıştırıcı kod ve veri arasında ayrım yapamaz.




Bir SQL ifadesi oluşturmak için dize birleştirme kullanmak, geliştiricilerin güvenliği dikkate almadığı eski uygulamalarda çok yaygındır. Sorun, bu kodlama tekniğinin ayrıştırıcıya ifadenin hangi kısmının kod hangi kısmının veri olduğunu söylememesidir. Kullanıcı girdisinin SQL ifadesine birleştirildiği durumlarda, bir saldırgan girdi verilerine SQL kodu ekleyerek SQL ifadesini değiştirebilir.

**1.** Güvenilmeyen girdi uygulama tarafından kabul edilebilir. Enjeksiyon güvenlik açığını azaltmanın çeşitli yolları vardır, beyaz liste, regex, vb. En iyi beş yol şunlardır. Derinlemesine savunma yaklaşımı için beşi birlikte kullanılmalıdır.

**1.** HtmlTüm kullanıcı girdilerini kodlayın.

**2.** Statik analiz araçlarını kullanma. Net, Java, python gibi diller için çoğu statik analiz doğrudur. Ancak JavaScript ve CSS'den enjeksiyon söz konusu olduğunda statik analiz bir sorun haline gelebilir.

**3.** SQL sorgularını parametrelendirin. SQL ayrıştırıcısının kod ve veri arasında ayrım yapabilmesi için deyimleri parametrelendiren programlama dili veya çerçevesi tarafından sağlanan SQL yöntemlerini kullanın.

**4.** Saklı Prosedürleri Kullanın. Saklı yordamlar genellikle SQL ayrıştırıcısının kod ve verileri birbirinden ayırmasına yardımcı olur. Ancak Saklı Yordamlar dinamik SQL ifadeleri oluşturmak için kullanılabilir ve kod ile verinin birbirine karışmasına neden olarak enjeksiyona karşı savunmasız hale gelmesine yol açabilir.

**5.** Güvenli kodlamaya yönelik en iyi uygulamalar için geliştirici eğitimi sağlayın.

### Kör SQL Enjeksiyonu

Tipik olarak SQL sorguları kullanıcıya sunulan arama sonuçlarını döndürür. Ancak, SQL sorgularının sayfanın nasıl oluşturulduğunu etkileyen perde arkasında gerçekleştiği durumlar vardır ve ne yazık ki saldırganlar çeşitli kullanıcı arayüzü öğelerinden gelen hata yanıtlarına dayanarak bilgi toplayabilir. Kör SQL enjeksiyonu, veritabanına doğru veya yanlış sorular soran ve uygulamanın yanıtına göre cevabı belirleyen bir saldırı türüdür.

Saldırgan, geçerli SQL için hangi hata yanıtlarının döndürüldüğünü ve geçersiz SQL için hangi yanıtların döndürüldüğünü belirlemek için SQL sorgularını kullanır. Daha sonra saldırgan araştırma yapabilir; örneğin "user_password_table" adında bir tablonun var olup olmadığını kontrol edebilir. Bu bilgiye sahip olduktan sonra, tabloyu kötü niyetle silmek için yukarıda açıklanan gibi bir saldırı kullanabilir veya tablodan bilgi döndürmeye çalışabilirler ("john" kullanıcı adı var mı?). Kör SQL enjeksiyonları hata mesajları yerine zamanlamaları da kullanabilir, örneğin geçersiz SQL'in yanıt vermesi 2 saniye sürüyorsa, ancak geçerli SQL 0,5 saniyede dönüyorsa, saldırgan bu bilgiyi kullanabilir.

### Parametrelendirilmiş SQL Sorguları

Parametrelendirilmiş SQL sorguları (bazen hazır deyimler olarak da adlandırılır) SQL sorgu dizesinin, istemci girdisinin SQL sözdiziminin bir parçası olarak ele alınamayacağı şekilde tanımlanmasına olanak tanır.
**Örnek 7.1'deki örneği ele alalım:**

**Örnek 7.1**

```
1    String query = "SELECT id, firstname, lastname FROM authors WHERE forename = ? and surname =
2    ?"; PreparedStatement pstmt = connection.prepareStatement( query );
3    pstmt.setString( 1, firstname );
4    pstmt.setString( 2, lastname );
```

Bu örnekte 'query' dizesi herhangi bir istemci girdisine dayanmayacak şekilde oluşturulmuş ve 'PreparedStatement' bu dizeden oluşturulmuştur. İstemci girdisi SQl'ye girileceği zaman, 'setString' fonksiyonu kullanılır ve ilk soru işareti "?" 'firstname' dize değeri ile değiştirilir, ikinci soru işareti ise 'lastname' değeri ile değiştirilir. 'setString' fonksiyonu çağrıldığında, bu fonksiyon otomatik olarak dize değeri içinde SQL sözdizimi bulunup bulunmadığını kontrol eder. Çoğu hazır deyim API'si, girilmesi gereken türü belirtmenize izin verir, örneğin 'setInt' veya 'setBinary' vb.

### Güvenli Dize Birleştirme?

Peki bu, DB işleme kodunuzda dize birleştirmeyi hiç kullanamayacağınız anlamına mı geliyor? Dize birleştirmeyi güvenli bir şekilde kullanmak mümkündür, ancak bir saldırgan uygulamanıza SQL sözdizimi enjekte etmeye çalışmasa bile hata riskini artırır.

İstemci girdi değeri ile birlikte asla dize birleştirme kullanmamalısınız. Hazırlanan deyimin SQL sorgusunu oluşturmak için "surname" istemci girdi değişkeninin varlığının (değerinin değil) kullanıldığı bir örneği ele alalım;

**Örnek 7.2**

```
1 String query = "Select id, firstname, lastname FROM authors WHERE forename = ?";
2 if (soyadı!= NULL && soyadı.length != 0) {
3    query += " and surname = ?";
4 }
5 query += ";";
6
7            PreparedStatement pstmt = connection.prepareStatement( query
8            ); pstmt.setString( 1, firstname);
9
10           if (soyadı!= NULL && soyadı.length != 0) { pstmt.setString( 2, soyadı ); }
```

Burada 'lastname' değeri kullanılmamakta, ancak var olup olmadığı değerlendirilmektedir. Ancak SQL ifadesi daha büyük olduğunda ve daha karmaşık iş mantığı içerdiğinde yine de bir risk vardır. Fonksiyonun ad veya soyadına göre arama yapacağı aşağıdaki örneği ele alalım:

**Örnek 7.3**

```
1 String query = "select id, firstname, lastname FROM authors";
2
3 if (( firstname != NULL && firstname.length != 0 ) && ( lastname != NULL && lastname.length != 0 )) {
4 query += "WHERE forename = ? AND surname = ?";
5 }
6 else if ( firstname != NULL && firstname.length != 0 ) {
7query              += "WHERE forename = ?";
8 }
9 else if ( soyadı!= NULL && soyadı.length != 0 ) {
       10query += "WHERE surname = ?";
11 }
12
13 query += ";";
14
15 PreparedStatement pstmt = connection.prepareStatement( query )
```

Bu mantık, firstname ya da lastname değerlerinden biri verildiğinde sorunsuz olacaktır, ancak ikisi de verilmediğinde SQL durumu herhangi bir WHERE cümlesine sahip olmayacak ve tüm tablo döndürülecektir. Bu bir SQL enjeksiyonu değildir (saldırgan iki değer geçmemek dışında bu duruma neden olacak hiçbir şey yapmamıştır) ancak sonuç aynıdır, parametrelendirilmiş bir sorgu kullanılmasına rağmen veritabanından bilgi sızdırılmıştır.

Bu nedenle, özellikle birleştirme işlemi where cümlesinde herhangi bir öğe oluşturmayı içeriyorsa, param eterize sorgular kullanılırken bile SQL sorgu dizeleri oluşturmak için dize birleştirme kullanmaktan kaçınılması önerilir.

### Esnek Parametreli İfadeleri Kullanma

İşlevsel gereksinimler genellikle çalıştırılan SQL sorgusunun kullanıcı girdisine göre esnek olmasını gerektirir, örneğin son kullanıcı işlem araması için bir zaman aralığı belirtirse bu kullanılmalıdır veya soyadına veya önadına ya da her ikisine göre sorgu yapmak isteyebilir. Bu durumda yukarıdaki güvenli dize birleştirme kullanılabilir, ancak

Bakım açısından bakıldığında bu, gelecekteki programcıların güvenli birleştirme ile güvenli olmayan sürüm (giriş dizesi değerlerini doğrudan kullanma) arasındaki farkı yanlış anlamalarına neden olabilir.

Esnek parametreli deyimler için bir seçenek, örneğin sağlanan girdi değerlerine göre doğru sorguyu seçmek için 'if' deyimlerini kullanmaktır:

**Örnek 7.4**

```
String sorgu;
PreparedStatement pstmt;

if ( (firstname!= NULL && firstname.length != 0)
   && lastname!= NULL && lastname.length != 0)
   ) {
   query ="Select id, firstname, lastname FROM authors WHERE forename = ? and surname =
   ?" pstmt = connection.prepareStatement( query );
   pstmt.setString( 1, firstname );
   pstmt.setString( 2, lastname );
}
else if (firstname != NULL && firstname.length != 0) {
   query ="Select id, firstname, lastname FROM authors WHERE forename = ?";
   pstmt = connection.prepareStatement( query );
   pstmt.setString( 1, firstname );
}
else if (soyadı != NULL && soyadı.length != 0){
   query ="Select id, firstname, lastname FROM authors WHERE surname= ?";
   pstmt = connection.prepareStatement( query );
   pstmt.setString( 1, soyadı);
}
else{
   throw NameNotSpecifiedException(); }
```

### PHP SQL Enjeksiyonu

Bir SQL enjeksiyon saldırısı, web uygulamasındaki istemci arayüzü aracılığıyla arka uç veritabanı sistemine SQL sorgu bölümlerinin enjekte edilmesinden oluşur. Bir SQL enjeksiyonunun başarılı bir şekilde istismar edilmesinin sonucu, sadece veri okumaktan verileri değiştirmeye veya sistem komutlarını çalıştırmaya kadar değişir. PHP'de SQL Enjeksiyonu bir numaralı saldırı vektörü ve aynı zamanda **örnek 7.5'te** gösterildiği gibi veri ihlallerinin bir numaralı nedeni olmaya devam etmektedir.

**Örnek 1 :**

**Örnek 7.5**

```
1 <?php
1 $pass=$_GET["pass"];
2 $con = mysql_connect('localhost', 'owasp', 'abc123');
3  mysql_select_db("owasp_php", $con);
4  $sql="SELECT card FROM users WHERE password = '".$pass."'";
5 $sonuç = mysql_query($sql);
6  ?>
```

PHP'de SQL Enjeksiyonunu önlemenin en yaygın yolları addslashes() ve mysql_ real_escape_string() gibi fonksiyonları kullanmaktır, ancak bu fonksiyonlar bazı durumlarda her zaman SQL Enjeksiyonuna neden olabilir.

**Eğik çizgi ekle :**
Sadece sorgu dizesini tırnak işaretleriyle sardığınız durumlarda addslashes() kullanarak Sql enjeksiyonunu önlersiniz. Aşağıdaki örnek yine de savunmasız olacaktır

**Örnek 7.6**

```
1        $id = addslashes( $_GET['id'] );
2        $query ='SELECT title FROM books WHERE id =' . $id;
```

**mysql_real_escape_string():**
mysql_real_escape_string() fonksiyonu addslashes() fonksiyonundan biraz daha güçlüdür, çünkü MySQL'in mysql_real_escape_string kütüphane fonksiyonunu çağırır ve aşağıdaki karakterlere ters eğik çizgi ekler: \x00, \n, \r, \, ', " ve \x1a.

addslashes() işlevinde olduğu gibi, mysql_real_escape_string() işlevi de yalnızca sorgu dizesi tırnak içine alınmışsa çalışacaktır. Aşağıdaki gibi bir d i z e yine de bir SQL enjeksiyonuna karşı savunmasız olacaktır:

SQL enjeksiyonları, bir web uygulamasına yapılan girdi arka uç veritabanına gönderilmeden önce kontrol edilmediğinde veya sterilize edilmediğinde ortaya çıkar.

Saldırgan, SQL komutlarını girdisinde geçirerek bu güvenlik açığından yararlanmaya çalışır ve böylece web uygulamasında programlanan yetkilendirme ve kimlik doğrulamasını atlayan bilgiler sağlamak gibi veritabanından istenmeyen bir yanıt oluşturur. Savunmasız java kodunun bir örneği örnek 7.7'de gösterilmektedir

**Örnek 7.7**

```
1  HttpServletRequest istek = ...;
2 String userName = request.getParameter("name");
3  Bağlantı con = ...
4   String query ="SELECT * FROM Users WHERE name ='" + kullanıcıAdı +"'";
5 con.execute(query);
```

**Savunmasız bir java kodu örneği**
"name" giriş parametresi, herhangi bir uygun doğrulama veya onaylama yapılmadan String sorgusuna aktarılır. 'SELECT* FROM users where name" is equal to the string 'username' sorgusu, sadece 'name'den farklı bir şeyi atlamak için kolayca kötüye kullanılabilir. Örneğin, saldırgan bu şekilde sadece belirli bir kullanıcıya ait kayıtları değil, tüm kullanıcı kayıtlarını ele geçirmeye çalışabilir

```
" VEYA 1=1.
```

### .NET Sql Enjeksiyonu

Framework 1.0 & 2.0, SQL enjeksiyonlarına karşı .NET'in sonraki sürümlerine göre daha savunmasız olabilir. MVC gibi ASP.NET'te zaten gömülü olan tasarım kalıplarının doğru şekilde uygulanması ve kullanılması sayesinde (sürüme de bağlı olarak), SQL enjeksiyonlarından arındırılmış uygulamalar oluşturmak mümkündür, ancak bir geliştiricinin SQL kodunu doğrudan kodda kullanmayı tercih edebileceği zamanlar olabilir.

**Örnek.**
Bir geliştirici, çalışanları **'ad'**, **'soyad'** ve '**kimlik'** alanlarında aramak için 3 alan ve gönder düğmesi içeren bir web sayfası oluşturur

Geliştirici, kodda örnek 7.8'deki gibi bir dize birleştirilmiş SQL deyimi veya saklı yordam uygular.

**Örnek 7.8**

```
1 SqlDataAdapter thisCommand = new SqlDataAdapter(
2 "SELECT ad, soyad FROM çalışanlar WHERE ei_id = '" +
    idNumber.Text + "'", thisConnection);
```

Bu kod, örnek 7.9'daki çalıştırılan SQL deyimine eşdeğerdir.

**Örnek 7.9**

```
1 SqlDataAdapter thisCommand = new SqlDataAdapter(
2 "SearchEmployeeSP '" + idNumber.Text + "'", thisConnection);
```

Bir bilgisayar korsanı daha sonra web arayüzü üzerinden aşağıdaki çalışan kimliğini girebilir "123';DROP TABLE pubs --" ve aşağıdaki kodu çalıştırabilir:
**SELECT name, lastname FROM authors WHERE ei_id = '123'; DROP TABLE pubs --'**

Noktalı virgül ";" SQL'e sql ifadesinin sonuna ulaştığına dair bir sinyal sağlar, ancak bilgisayar korsanı bunu kötü amaçlı SQL koduyla ifadeye devam etmek için kullanır
**; DROP TABLE pubs;**

### Parametre koleksiyonları

SqlParameterCollection gibi parametre koleksiyonları, tür denetimi ve uzunluk doğrulaması sağlar. Bir parametre koleksiyonu kullanırsanız, girdi değişmez bir değer olarak değerlendirilir ve SQL Server bunu çalıştırılabilir kod olarak değerlendirmez ve bu nedenle yük enjekte edilemez.

Parametreler koleksiyonu kullanmak, tür ve uzunluk kontrollerini uygulamanızı sağlar. Aralığın dışındaki değerler bir istisnayı tetikler. İstisnayı doğru şekilde ele aldığınızdan emin olun. SqlParameterCollection örneği:
**Hibernate Sorgu Dili (HQL)**

**Örnek 7.10**

```
1 using (SqlConnection conn = new SqlConnection(connectionString)) {
2 DataSet dataObj = new DataSet();
3 SqlDataAdapter sqlAdapter = new SqlDataAdapter("StoredProc", conn); sqlAdapter.SelectCommand.
CommandType =
4 CommandType.StoredProcedure;
5 sqlAdapter.SelectCommand.Parameters.Add("@usrId", SqlDbType.VarChar, 15);
6  sqlAdapter.SelectCommand.Parameters["@usrId "].Value = UID.Text;
```

Hibernate, Nesne / İlişkisel Eşleme (ORM) aracılığıyla Java etki alanı nesnelerinin depolanmasını ve alınmasını kolaylaştırır.

Hibernate gibi ORM çözümlerinin SQL Enjeksiyonuna karşı dayanıklı olduğu çok yaygın bir yanılgıdır. Hibernate "yerel SQL" kullanımına izin verir ve HQL (Hibernate Sorgu Dili) adı verilen tescilli bir sorgu dili tanımlar; ilki SQL Enjeksiyonuna, ikincisi ise HQL (veya ORM) enjeksiyonuna eğilimlidir.

**Ne İncelenmeli**

• Tür, uzunluk, biçim ve aralığı test ederek kullanıcı girdisini her zaman doğrulayın.

• Girdinin boyutunu ve veri türünü test edin ve uygun sınırları uygulayın.

• Dize değişkenlerinin içeriğini test edin ve yalnızca beklenen değerleri kabul edin. İkili veri, kaçış dizileri ve yorum karakterleri içeren girişleri reddedin.

• XML belgeleriyle çalışırken, girilen tüm verileri şemasına göre doğrulayın.

• SQL ifadelerini asla doğrudan kullanıcı girdisinden oluşturmayın.

• Kullanıcı girdisini doğrulamak için saklı yordamları kullanın, saklı yordamları kullanmadığınızda platform tarafından sağlanan SQL API'sini kullanın. yani Parametrelendirilmiş İfadeler.

• Birden fazla doğrulama katmanı uygulayın.

• Doğrulanmamış kullanıcı girdilerini asla birleştirmeyin. Dize birleştirme, komut dosyası ekleme için birincil giriş noktasıdır.

EXECUTE, EXEC, dış kaynakları veya komut satırını çağırabilen tüm SQL çağrılarını çağıran tüm kodları gözden geçirmelisiniz.

**OWASP Referansları**

• **https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet OWASP SQL Enjeksiyonu Önleme Hile Sayfası**

• **https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet OWASP Sorgu Parametrelendirme Hile Sayfası**

• **https://www.owasp.org/index.php/Command_Injection OWASP Komut Enjeksiyonu Makalesi**

• **https://www.owasp.org/index.php/XXE OWASP XML eXternal Entity (XXE) Referans Makalesi**

• https://www.owasp.org/index.php/ASVS ASVS: Çıktı Kodlama/Çıkış Gereksinimleri (V6)

• https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005) OWASP Test Kılavuzu: SQL Enjeksiyon Testi Bölümü

**Dış Referanslar**
• http://cwe.mitre.org/data/definitions/77.html Komut Enjeksiyonunda CWE Girişi 77

• http://cwe.mitre.org/data/definitions/89.html SQL Enjeksiyonunda CWE Girişi 89

• http://cwe.mitre.org/data/definitions/564.html **CWE Entry 564 on Hibernate Injection**

• **Livshits ve Lam, 2005 "Java Uygulamalarındaki Güvenlik Açıklarını Statik Analiz ile Bulma"**
https://www.usenix.org/legacy/event/sec05/tech/full_papers/livshits/livshits_html/#sec:sqlinjexample
**adresinde mevcuttur**

• http://www.php.net/manual/en/book.pdo.php PDO

• https://technet.microsoft.com/en-us/library/ms161953(v=sql.105).aspx

### 7.3 JSON (JavaScript Nesne Gösterimi)

JSON (JavaScript Object Notation), bir sunucu ile web uygulamaları arasında veri iletmek için okunması kolay metin kullanan açık standart bir formattır. JSON verileri çok sayıda programlama dili tarafından kullanılabilir ve XML'in yerini alan fiili standart haline gelmektedir.

JSON ana güvenlik endişesi, JavaScript'e dinamik olarak gömülü JSON metnidir, çünkü bu enjeksiyon çok gerçek bir güvenlik açığıdır. Programdaki güvenlik açığı, yanlışlıkla kötü amaçlı bir komut dosyası çalıştırabilir veya kötü amaçlı komut dosyasını bir veritabanına depolayabilir. İnternetten alınan verilerle uğraşırken bu çok gerçek bir olasılıktır.

Kodu gözden geçiren kişinin JSON'un Javascript eval ile kullanılmadığından emin olması gerekir. JSON.parse(...)

kullanıldığından emin olun. Var parsed_object = eval("(" + Jason_text + ")"); // Kodu gözden geçiren kişi için

kırmızı bayrak.
JSON.parse(text[, reviver]); ... // Javascript eval fonksiyonunu kullanmaktan çok daha iyi.

Kodu gözden geçiren kişi, geliştiricinin metin/dize verilerinde bilinen kötü kalıpları reddetmeye çalışmadığından emin olmalıdır, regex veya diğer araçları kullanmak hatalarla doludur ve doğruluk testini çok zorlaştırır. Yalnızca beyaz listeye alınmış alfanümerik anahtar kelimelere ve dikkatlice doğrulanmış sayılara izin verin.

JSON verilerinin dinamik HTML oluşturmasına izin vermeyin. Her zaman innerText veya CreateText- Node(...) gibi güvenli DOM özelliklerini kullanın.

### Nesne / İlişkisel Eşleme (ORM)

Nesne/Relasyon Eşleme (ORM), HQL (Hibernate Query Language) veya .NET Entity framework aracılığıyla etki alanı nesnelerinin depolanmasını ve alınmasını kolaylaştırır.

Hibernate gibi ORM çözümlerinin SQL Enjeksiyonuna dayanıklı olduğu çok yaygın bir yanılgıdır. Öyle değildirler. ORM'ler "yerel SQL" kullanımına izin verir. HQL adı verilen tescilli sorgu dili SQL Enjeksiyonuna eğilimlidir ve daha sonra HQL (veya ORM) enjeksiyonuna eğilimlidir. Linq SQL değildir ve bu nedenle SQL enjeksiyonuna eğilimli

değildir. Ancak linq üzerinden excutequery veya excutecommand kullanmak programın linq koruma mekanizmasını kullanmamasına ve SQL enjeksiyonuna açık olmasına neden olur.

**Kötü Java Kodu Örnekleri**
Liste sonuçları = session.createQuery("from Items as item where item.id =" + currentItem.getId()).list();

NHibernate, Microsoft .NET platformu için bir ORM çözümü olması dışında Hibernate ile aynıdır. NHibernate, dinamik sorgularım kullanıldığında SQL enjeksiyonuna karşı da savunmasızdır.

**Kötü .Net Kod Örneği**

```
string userName = ctx.GetAuthenticatedUserName();
 String query = "SELECT * FROM Items WHERE owner = '"
 + userName + "' AND itemname = '"
 + ItemName.Text + "'";
 Liste öğeleri = sess.CreateSQLQuery(query).List()
```

**Kod Gözden Geçirme Eylemi**
Kodu gözden geçiren kişinin, HQL sorgusunda kullanılan verilerin kod olarak değil veri olarak kullanılabilmesi için HQL parametrelendirilmiş sorguları kullandığından emin olması gerekir. Ayrıca **https://docs.jboss.org/hibernate/ orm/3.3/reference/en/html/querycriteria.html** adresindeki Kriter API'sini de kullanabilirler.

### 7.5 İçerik Güvenliği Politikası (CSP)
İstemci tarayıcısına hangi konumdan ve/veya hangi tür kaynakların yüklenmesine izin verileceğini bildirme imkanı sunan bir W3C spesifikasyonudur. Bir yükleme davranışını tanımlamak için CSP spesifikasyonu, bir direktifin bir hedef kaynak türü için bir yükleme davranışını tanımladığı "direktif" kullanır. CSP, Cross Site Scripting (XSS) ve veri enjeksiyon saldırıları da dahil olmak üzere belirli saldırı türlerini tespit etmeye ve azaltmaya yardımcı olur. Bu saldırılar veri hırsızlığından site tahrifatına veya kötü amaçlı yazılım dağıtımına kadar her şey için kullanılır

Yönergeler HTTP yanıt başlığı (bir sunucu belirli bir kaynak gösterimiyle birden fazla CSP HTTP başlık alanı gönderebilir ve bir sunucu aynı kaynağın farklı gösterimleriyle veya farklı kaynaklarla farklı CSP başlık alanı değerleri gönderebilir) veya HTML Meta etiketi kullanılarak belirtilebilir, aşağıdaki HTTP başlıkları teknik özellikler tarafından tanımlanmıştır:

- Content-Security-Policy : W3C Specs tarafından standart başlık olarak tanımlanmıştır, Chrome sürüm 25 ve sonrası, Firefox sürüm 23 ve sonrası, Opera sürüm 19 ve sonrası tarafından kullanılır.

- X-Content-Security-Policy : Sürüm 23'e kadar Firefox ve Internet Explorer sürüm 10 (İçerik Güvenliği İlkesini kısmen uygulayan) tarafından kullanılır.

- X-WebKit-CSP : Sürüm 25'e kadar Chrome tarafından kullanılır

**Risk**
CSP ile ilgili riskin 2 ana kaynağı olabilir:
- Politikaların yanlış yapılandırılması,
- Çok müsamahakâr politikalar.

**Ne İncelenmeli**
Kodu gözden geçiren kişinin, uygulama tasarımı için hangi içerik güvenliği politikalarının gerekli olduğunu ve bu politikaların uygulama tarafından kullanıldığından emin olmak için nasıl test edildiğini anlaması gerekir.

**Güvenlikle ilgili faydalı HTTP üstbilgileri**

Çoğu mimaride bu başlıklar, gerçek uygulamanın kodunu değiştirmeden web sunucularının yapılandırmasında ayarlanabilir. Bu, mevcut sorunların en azından kısmen azaltılması için önemli ölçüde daha hızlı ve daha ucuz bir yöntem ve yeni uygulamalar için ek bir savunma katmanı sunar.

**Tablo 11:** Güvenlikle İlgili HTTP Üstbilgileri

| Başlık adı | Açıklama | Örnek |
|---|---|---|
| Strict-Transport-Security https://tools.ietf.org/html/rfc6797 | HTTP Strict-Transport-Security (HSTS) sunucuya güvenli (SSL/TLS üzerinden HTTP) bağlantıları zorunlu kılar. Bu, çerezler ve harici bağlantılar aracılığıyla oturum verilerini sızdıran web uygulamalarındaki hataların etkisini azaltır ve ortadaki adam saldırılarına karşı savunma sağlar. HSTS ayrıca kullanıcıların SSL anlaşma uyarılarını göz ardı etmesini de engeller. | Strict-Transport-Security: max-age=16070400; includeSubDomains |
| X-Frame-Options https://tools.ietf.org/html/draft-ietf-websec-x-frame-options-01<br><br>Çerçeve Seçenekleri https://tools.ietf.org/html/draft-ietf-websec-frame-options-00 | Click jacking koruması sağlar. Değerler: deny - bir çerçeve içinde işleme yok, sameorigin - kaynak uyuşmazlığı varsa işleme yok, allow-from: DOMAIN - DOMAIN'den yüklenen çerçeve tarafından çerçevelenirse işleme izin ver | X-Frame-Options: deny |
| X-XSS Koruması<br><br>[http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx X-XSS-Protection] | Bu başlık, en yeni web tarayıcılarında yerleşik olarak bulunan Siteler arası komut dosyası oluşturma (XSS) filtresini etkinleştirir. Genellikle varsayılan olarak zaten etkindir, bu nedenle bu başlığın rolü, kullanıcı tarafından devre dışı bırakılmışsa bu belirli web sitesi için filtreyi yeniden etkinleştirmektir. Bu başlık IE 8+ ve Chrome'da desteklenmektedir (hangi sürümler olduğundan emin değilim). Anti-XSS filtresi Chrome'a eklendi<br>4. Bu sürümün bu başlığı onurlandırıp onurlandırmadığı bilinmiyor. | X-XSS-Protection: 1; mode=block |
| X-Content-Type-Options<br><br>https://blogs.msdn.microsoft.com/ie/2008/09/02/ie8-security-part-vi-beta-2-update/ | Tanımlanan tek değer olan "nosniff", Internet Explorer ve Google Chrome'un bildirilen içerik türünden farklı bir yanıtı MIME koklamasını engeller. Bu, uzantıları indirirken Google Chrome için de geçerlidir. Bu, drive-by down-load saldırılarına ve akıllıca bir adlandırma ile MSIE tarafından çalıştırılabilir veya dinamik HTML dosyaları olarak değerlendirilebilecek kullanıcı tarafından yüklenen içerik sunan sitelere maruz kalmayı azaltır. | X-Content-Type-Options: nosniff |
| Content-Security-Policy, X-Content-Security-policy,X-WebKit-CSP<br><br>https://www.w3.org/TR/CSP/ | İçerik Güvenliği Politikası dikkatli bir ayarlama ve politikanın kesin olarak tanımlanmasını gerektirir. Etkinleştirilirse, CSP'nin tarayıcının sayfaları oluşturma şekli üzerinde önemli bir etkisi vardır (örneğin, satır içi JavaS-cript varsayılan olarak devre dışıdır ve politikada açıkça izin verilmelidir). CSP, siteler arası komut dosyası oluşturma ve diğer siteler arası enjeksiyonlar dahil olmak üzere çok çeşitli saldırıları önler. | Content-Security-Policy: default-src 'self' |
| Content-Security-Policy-Report_Only<br><br>https://www.w3.org/TR/CSP/ | Content-Security-Policy gibi, ancak yalnızca raporlar. Uygulama, ayarlama ve test çalışmaları sırasında faydalıdır. | Content-Security-Policy-Report-Only: de- fault-src 'self'; report-uri http://loghost.example.com/reports.jsp |

Spring Security kütüphanesinin bu başlıklar konusunda yardımcı olabileceğini unutmayın, bkz. http://docs.spring.io/spring-security/site/ docs/current/reference/html/headers.html

**Referanslar**
**Apache:** http://httpd.apache.org/docs/2.0/mod/mod_headers.html
**IIS:** http://technet.microsoft.com/pl-pl/library/cc753133(v=ws.10).aspx

### 7.6 Girdi Doğrulama

Girdi doğrulama, uygulama güvenliği için en etkili teknik kontrollerden biridir. Siteler arası komut dosyası oluşturma, çeşitli enjeksiyon biçimleri ve bazı arabellek taşmaları dahil olmak üzere çok sayıda güvenlik açığını azaltabilir. Girdi doğrulama, form alanı değerlerini kontrol etmekten daha fazlasıdır.

Kullanıcılardan gelen tüm verilerin güvenilmez olarak kabul edilmesi gerekir. Güvenli kodlamanın en önemli kurallarından birinin "Kullanıcı girdisine güvenmeyin" olduğunu unutmayın. Kullanıcı verilerini her zaman uygulamanızın neyi başarmaya çalıştığını tam olarak bilerek doğrulayın.

Düzenli ifadeler kullanıcı girdisini doğrulamak için kullanılabilir, ancak düzenli ifadeler ne kadar karmaşık olursa, tam kanıt olmama ve köşe durumları için hata içerme olasılığı o kadar artar. Düzenli ifadelerin QA tarafından test edilmesi de çok zordur. Düzenli ifadeler, kodu gözden geçiren kişinin düzenli ifadeler üzerinde iyi bir inceleme yapmasını da zorlaştırabilir.

**Veri Doğrulama**
Sisteme (ve sistemler/uygulamalar arasındaki) tüm harici girdiler girdi doğrulamasından geçmelidir. Doğrulama kuralları uygulama için iş gereksinimleri tarafından tanımlanır. Mümkünse, bir tam eşleşme doğrulayıcısı uygulanmalıdır. Tam eşleşme yalnızca beklenen bir değere uyan verilere izin verir. Biraz daha zayıf ancak daha esnek olan "Bilinen iyi" yaklaşımı (beyaz liste) yaygındır. Bilinen iyi yalnızca beyaz liste içinde tanımlanan karakterlere/ASCII aralıklarına izin verir.

Böyle bir aralık, girdi alanının iş gereksinimleri tarafından tanımlanır. Veri doğrulamaya yönelik diğer yaklaşımlar "kötü karakterlerin" kara listesi olan "bilinen kötü" yaklaşımıdır. Bu yaklaşım geleceğe dönük değildir ve bakım gerektirecektir. "Kötü karakterleri kodla" çok zayıf bir yaklaşımdır, çünkü sadece "kötü" olarak kabul edilen karakterleri uygulamanın işlevselliğini etkilemeyecek bir formata kodlar.

**İş Doğrulama**
İş doğrulaması iş mantığı ile ilgilidir. Bu mantığı gerçekleştiren kodu incelemeden önce iş mantığının anlaşılması gerekir. İş doğrulaması, bir kullanıcı tarafından girilen değer aralığını veya bir işlemi sınırlamak ya da iş açısından çok fazla anlam ifade etmeyen girdileri reddetmek için kullanılabilir. İş doğrulaması için kodun gözden geçirilmesi, tamsayı taşmaları gibi sorunlara yol açabilecek yuvarlama hatalarını veya kayan nokta sorunlarını da içerebilir, bu da alt satıra önemli ölçüde zarar verebilir.

**Kanonikleştirme**
Kanonikleştirme, bir ismin çeşitli eşdeğer biçimlerinin tek bir standart isme veya "kanonik" isme çözümlenebildiği süreçtir.

En popüler kodlamalar UTF-8, UTF-16 ve benzerleridir (RFC 2279'da ayrıntılı olarak açıklanmıştır). Nokta/nokta (.) gibi tek bir karakter birçok farklı şekilde temsil edilebilir: ASCII 2E, Unicode C0 AE ve diğerleri.

Kullanıcı girdisini kodlamanın sayısız yolu olduğundan, bir web uygulamasının filtreleri dikkatli bir şekilde oluşturulmazsa kolayca atlatılabilir.

**Kötü Örnek**

**Örnek 7.11**

```
public static void main(String[] args)
  { File x = new File("/cmd/" +
  args[1]); String absPath =
  x.getAbsolutePath();
}
```

**İyi Örnek**

**Örnek 7.12**

```
public static void main(String[] args) throws IOException {
   Dosya x = new Dosya("/cmd/" + args[1]);
   String canonicalPath = x.getCanonicalPath();
```

**.NET İstek Doğrulama**
Bir çözüm .Net "İstek Doğrulama" kullanmaktır. İstek doğrulamayı kullanmak, kullanıcı verilerini doğrulamak için iyi bir başlangıçtır ve kullanışlıdır. Dezavantajı çok geneldir ve kullanıcı verilerine tam güven sağlamak için tüm gereksinimlerimizi karşılayacak kadar spesifik değildir.

Uygulamanızı siteler arası komut dosyası saldırılarına karşı korumak için istek doğrulamayı asla kullanamazsınız.

Aşağıdaki örnekte, bir kullanıcı tarafından sağlanan Uri'nin geçerli olup olmadığını belirlemek için Uri sınıfındaki statik bir yöntemin nasıl kullanılacağı gösterilmektedir.

```
var isValidUri = Uri.IsWellFormedUriString(passedUri, UriKind.Absolute);
```

Ancak, Uri'yi yeterince doğrulamak için, http veya https belirttiğinden emin olmak için de kontrol etmelisiniz. Aşağıdaki örnek, Uri'nin geçerli olduğunu doğrulamak için örnek yöntemleri kullanır.

```
var uriToVerify = new Uri(passedUri);
var isValidUri = uriToVerify.IsWellFormedOriginalString();
var isValidScheme = uriToVerify.Scheme == "http" || uriToVerify.Scheme == "https";
```

Kullanıcı girdisini HTML olarak oluşturmadan veya kullanıcı girdisini bir SQL sorgusuna dahil etmeden önce, zararlı kodun dahil edilmediğinden emin olmak için değerleri kodlayın.

Değeri işaretlemede <% ile HTML kodlayabilirsiniz: %> sözdizimiyle aşağıda gösterildiği gibi kodlayabilirsiniz.

```
<span><%: userInput %></span>
```

Veya Razor sözdiziminde, aşağıda gösterildiği gibi @ ile HTML kodlaması yapabilirsiniz.

```
<span>@userInput</span>
```

Bir sonraki örnek, code-behind içinde bir değerin HTML ile nasıl kodlanacağını göstermektedir.

```
var encodedInput = Server.HtmlEncode(userInput);
```

**Yönetilen Kod ve Yönetilmeyen Kod**
Hem Java hem de .Net yönetilen ve yönetilmeyen kod kavramına sahiptir. Yerel kodun çağrılması sırasında bu korumalardan bazılarını sunmak için, yerel bir yöntemi genel olarak bildirmeyin. Bunun yerine, özel olarak bildirin ve işlevselliği genel bir sarmalayıcı yöntem aracılığıyla ortaya çıkarın. Sarmalayıcı, yerel yöntemin çağrılmasından önce gerekli girdi doğrulamasını güvenli bir şekilde gerçekleştirebilir:

Veri Doğrulama özelliğine sahip bir Yerel Yöntemi çağırmak için Java Örnek kodu

**Örnek 7.13**

```
public final class NativeMethodWrapper {
                private native void nativeOperation(byte[] data, int offset, int len);


                public void doOperation(byte[] data, int offset, int len) {
                  // değiştirilebilir girdiyi
                  kopyala data =
                  data.clone();

                  // girdiyi doğrula
                  // Ofset+len'in tamsayı taşmasına maruz kalacağını unutmayın.
                  // Örneğin offset = 1 ve len = Integer.MAX_VALUE ise,
                  // o zaman offset+len == Integer.MIN_VALUE daha düşüktür
                  // data.length'den daha fazla.
                  // Daha ileri,
                  // formundaki döngüler
                  //     for (int i=offset; i<offset+len; ++i) { ... }
                  // bir istisna fırlatmaz veya yerel kodun
                  // crash.

                  if (offset < 0 || len < 0 || offset > data.length - len) {
                      throw new IllegalArgumentException();
                  }

                  nativeOperation(data, offset, len);
                }
              }
```

**Kod İnceleme Görevlisi için veri doğrulamaları kontrol listesi.**
- Bir Veri Doğrulama mekanizmasının mevcut olduğundan emin olun.

- HTTP üstbilgileri, giriş alanları, gizli girişler gibi kötü niyetli bir kullanıcı tarafından değiştirilebilecek (ve değiştirilecek) tüm girdilerin

alanları, açılır listeler ve diğer web bileşenleri uygun şekilde doğrulanır.

- Tüm girişlerde uygun uzunluk kontrollerinin mevcut olduğundan emin olun.

- Tüm alanların, çerezlerin, http başlıklarının/bağlantılarının ve form alanlarının doğrulandığından emin olun.

- Verilerin iyi biçimlendirildiğinden ve mümkünse yalnızca bilinen iyi karakterler içerdiğinden emin olun.

- Veri doğrulamanın sunucu tarafında gerçekleştiğinden emin olun.

- Veri doğrulamanın nerede gerçekleştiğini ve merkezi bir model mi yoksa merkezi olmayan bir model mi kullanıldığını inceleyin.

- Veri doğrulama modelinde arka kapı olmadığından emin olun.

- "Altın Kural: Ne olursa olsun tüm dış girdiler incelenecek ve onaylanacaktır."

**Kaynaklar:**
http://msdn.microsoft.com/en-us/library/vstudio/system.uri


A2


# KIRIK KIMLIK DOĞRULAMA VE OTURUM YÖNETIMI

## 8.1 Genel Bakış

Web uygulamaları ve Web hizmetlerinin her ikisi de kullanıcı kimliği ve parolalar aracılığıyla oturum açma işlemlerinden erişim kontrolünün birincil aracı olarak kimlik doğrulamayı kullanır. Bu kontrol, gizli dosyaların, verilerin veya web sayfalarının bilgisayar korsanları veya gerekli erişim kontrol seviyesine sahip olmayan kullanıcılar tarafından erişilmesini önlemek için çok önemlidir.

## 8.2 Açıklama

Kimlik doğrulama, korumak istediğiniz işlevselliğe açılan kapı olduğu için önemlidir. Bir kullanıcının kimliği doğrulandıktan sonra, istekleri, kimliği doğrulanmamış kullanıcıların engelleneceği uygulamanızla belirli düzeyde etkileşim gerçekleştirme yetkisine sahip olacaktır. Kullanıcıların kimlik doğrulama bilgilerini veya belirteçlerini nasıl yönettiklerini kontrol edemezsiniz, ancak uygun kimlik doğrulama gerçekleşmeden uygulama işlevlerini gerçekleştirmenin artık bir yolu olmadığından emin olabilirsiniz.

Şifrelerin en yaygın olduğu birçok kimlik doğrulama biçimi vardır. Diğer formlar arasında istemci sertifikaları, biyometri, SMS veya özel cihazlar üzerinden tek seferlik şifreler veya Açık Yetkilendirme (OAUTH) veya Tek Oturum Açma (SSO) gibi kimlik doğrulama çerçeveleri yer alır.

Tipik olarak kimlik doğrulama, kullanıcı bir web sitesine giriş yaptığında bir kez yapılır ve başarılı kimlik doğrulama, kullanıcı için bir web oturumunun kurulmasıyla sonuçlanır (bkz. Oturum Yönetimi). Kullanıcı yüksek riskli bir işlevi yerine getirmeye çalışırsa daha sonra daha fazla (ve daha güçlü) kimlik doğrulama talep edilebilir, örneğin bir banka kullanıcısından para transferine izin vermeden önce kayıtlı telefon numarasına gönderilen 6 haneli bir numarayı onaylaması istenebilir.

Kimlik doğrulama, bir şirketin güvenlik duvarı içinde olduğu kadar dışında da önemlidir. Saldırganlar, sırf güvenlik duvarından geçmenin bir yolunu buldukları için bir şirketin dahili uygulamalarında serbestçe dolaşamamalıdır. Ayrıca ayrıcalıkların (veya görevlerin) ayrılması, hesaplarda çalışan birinin bir depodaki kodu değiştirememesi veya uygulama yöneticilerinin bordro tablolarını düzenleyememesi gerektiği anlamına gelir.

## 8.3 Ne İncelenmeli

Kimlik doğrulama işlevlerini yerine getiren kod modüllerini incelerken dikkat edilmesi gereken bazı yaygın sorunlar şunlardır:

- Oturum açma sayfasının yalnızca TLS üzerinden kullanılabilir olduğundan emin olun. Bazı siteler oturum açma sayfasını HTTP olarak bırakır, ancak form gönderme URL'sini HTTPS yapar, böylece kullanıcıların kullanıcı adı ve şifresi sunucuya gönderilirken şifrelenir. Ancak oturum açma sayfası güvenli değilse, ortadaki adamın form gönderme URL'sini HTTP URL'si olarak değiştirme riski vardır ve kullanıcı kullanıcı adı ve parolasını girdiğinde bunlar açık olarak gönderilir.

- Kullanıcı adlarınızın/kullanıcı kimliklerinizin büyük/küçük harfe duyarlı olmadığından emin olun. Birçok site kullanıcı adları için e-posta adreslerini kullanır ve e-posta adresleri zaten büyük/küçük harfe duyarlı değildir. Ne olursa olsun, 'smith' kullanıcısı ile 'Smith' kullanıcısının farklı kullanıcılar olması çok garip olurdu. Ciddi bir karışıklığa neden olabilir.

- Geçersiz kullanıcı adları veya parolalar için hata mesajlarının bilgi sızdırmadığından emin olun. Hata mesajı kullanıcı adının geçerli olduğunu ancak parolanın yanlış olduğunu gösteriyorsa, saldırganlar bu kullanıcı adının var olduğunu bilecektir. Parola yanlışsa, nasıl yanlış olduğunu belirtmeyin.

- Kullanıcının yazdığı her karakterin gerçekten parolaya dahil edildiğinden emin olun.

• Geçersiz şifreleri kaydetmeyin. Çoğu zaman kullanıcı adı olarak bir e-posta adresi kullanılır ve bu kullanıcıların ezberledikleri birkaç şifre vardır ancak web sitenizde hangisini kullandıklarını unutabilirler. İlk kez siteniz için geçersiz, ancak bu kullanıcının (kullanıcı adıyla tanımlanan) diğer birçok site için geçerli bir parola kullanabilirler. Eğer bu kullanıcı adı ve şifre kombinasyonunu kaydederseniz ve bu kayıt dışarı sızarsa, sitenizdeki bu düşük seviyeli komplo diğer birçok siteyi olumsuz etkileyebilir.

• Daha uzun parolalar daha fazla karakter kombinasyonu sağlar ve sonuç olarak bir saldırganın tahmin etmesini daha zor hale getirir. Parolaların minimum uzunluğu uygulama tarafından zorunlu kılınmalıdır. Parolaların 10 karakterden kısa olması zayıf olarak kabul edilir. Parolalar teşvik edilmelidir. Parola uzunlukları hakkında daha fazla bilgi için OWASP Authentication Cheat Sheet'e bakınız.

• Kaba kuvvet saldırılarını önlemek için geçici hesap kilitlemeleri uygulayın veya oturum açma yanıtlarını sınırlandırın. Bir kullanıcı doğru kullanıcı adı ve parolayı 5 kez giremezse, hesabı X dakika boyunca kilitleyin veya oturum açma yanıtlarının fazladan 10 saniye sürdüğü bir mantık uygulayın. Ancak dikkatli olun, bu durum kullanıcı adının geçerli olduğu gerçeğini sürekli rastgele kullanıcı adları deneyen saldırganlara sızdırabilir, bu nedenle ekstra bir önlem olarak geçersiz kullanıcı adları için de aynı mantığı uygulamayı düşünün.

• Dahili sistemler için, kullanıcıları belirli bir süre sonra parolalarını değiştirmeye zorlamayı düşünün ve kullanıcının eski parolasını tekrar kullanmadığından emin olmak için son 5 veya daha fazla parolanın bir referansını (örneğin hash) saklayın.

• Parola karmaşıklığı, kullanıcıların çeşitli karakter türlerini (ör. büyük ve küçük harfler, sayılar, noktalama işaretleri, vb.) içeren parola dizeleri seçmeleri sağlanarak uygulanmalıdır. İdeal olarak, uygulama kullanıcıya yeni parolasını yazarken yeni parolasının karmaşıklık ilkesinin ne kadarını karşıladığını gösterecektir. Parola karmaşıklığı hakkında daha fazla bilgi için OWASP Authentication Cheat Sheet'e bakın.

• Bir uygulamanın, bir kullanıcının şifresini unutması durumunda hesabına erişim sağlaması için bir araç sağlayan bir mekanizmaya sahip olması yaygındır. Bu, kimliği doğrulanmamış kullanıcılar (parolalarını girmedikleri için) tarafından çağrılan bir web sitesi işlevselliği örneğidir. Bu gibi arayüzlerin kötüye kullanıma karşı korunduğundan emin olun, eğer şifre hatırlatma talebi kayıtlı kullanıcıya bir e-posta veya SMS gönderilmesiyle sonuçlanıyorsa, şifre sıfırlama özelliğinin saldırganlar tarafından sürekli olarak kullanıcı adını bu forma girerek kullanıcıya spam göndermek için kullanılmasına izin vermeyin. Bu özelliğe ilişkin ayrıntılar için lütfen Şifremi Unuttum Hile Sayfası'na bakın.

• Bir uygulamanın doğru kriptografik tekniği kullanarak parola saklaması çok önemlidir. Bu özellik hakkında ayrıntılı bilgi için lütfen Parola Saklama Hile Sayfası'na bakın.

MVC .NET'i incelerken, uygulamanın güvenli bir bağlantı üzerinden iletildiğinden ve alındığından emin olmak önemlidir. Sadece oturum açma sayfalarının değil, tüm web sayfalarının SSL kullanması önerilir. Kullanıcıların kimlik bilgileri olarak yararlı olan oturum çerezlerini korumamız gerekir.

**Örnek 8.1**

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters) {
" filters.Add(new RequireHttpsAttribute()); "
}
```

global.asax dosyasında RegisterGlobalFilters yöntemini inceleyebiliriz.

RequireHttpsAttribute() niteliği, uygulamanın SSL/TLS üzerinden çalıştığından emin olmak için kullanılabilir. SSL/TLS siteleri için bunun etkinleştirilmesi önerilir.

• Bankacılık işlemleri, kullanıcı profili güncellemeleri vb. gibi yüksek riskli işlevler için çok faktörlü kimlik doğrulama (MFA) kullanın. Bu aynı zamanda CSRF ve oturum ele geçirme saldırılarına karşı da koruma sağlar. MFA, oturum açmak veya bir işlemi gerçekleştirmek için birden fazla kimlik doğrulama faktörü kullanmaktır:
• Bildiğiniz bir şey (hesap bilgileri veya şifreler)
• Sahip olduğunuz bir şey (jetonlar veya cep telefonları)
• Olduğunuz bir şey (biyometri)

• İstemci makine kontrollü bir ortamdaysa, iki yönlü kimlik doğrulama olarak da bilinen SSL İstemci Kimlik Doğrulamasını kullanın. SSL kimlik doğrulaması, hem tarayıcı hem de sunucunun TLS el sıkışma işlemi sırasında kendi SSL sertifikalarını göndermesinden oluşur. Sunucu yöneticisi istemci sertifikalarını oluşturup yayınlayabildiği için bu daha güçlü kimlik doğrulama sağlar ve sunucunun yalnızca bu istemci SSL sertifikasının yüklü olduğu makinelerden gelen oturum açma isteklerine güvenmesine olanak tanır. İstemci sertifikasının güvenli bir şekilde iletilmesi önemlidir.

### Referanslar
• https://www.owasp.org/index.php/Authentication_Cheat_Sheet
• http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf
• http://www.codeproject.com/Articles/326574/An-Introduction-to-Mutual-SSL-Authentication
• https://cwe.mitre.org/data/definitions/287.html Uygunsuz Kimlik Doğrulama
• **Kimlik Doğrulama için OWASP ASVS gereksinim alanları (V2)**

## 8.4 Şifremi Unuttum
**Genel Bakış**
Web sitenizin kullanıcı kimlik doğrulamasına sahip olması gerekiyorsa, büyük olasılıkla kullanıcı erişimlerini doğrulamak için kullanıcı adı ve parola gerektirecektir. Ancak bilgisayar sistemlerinin karmaşıklığı arttıkça kullanıcıların kimliklerinin doğrulanması da artmıştır. Sonuç olarak, kodu gözden geçiren kişinin bu bölümde "Doğrudan Kimlik Doğrulama" modeli olarak adlandırılan kullanıcı kimlik doğrulamasının faydalarının ve dezavantajlarının farkında olması gerekir. Bu bölümde, kullanıcıların kullanıcı kimliğini ve/veya şifresini unuttuğu durumlara yönelik tasarım modelleri ve kullanıcı tarafından unutulduğunda kullanıcı kimliği ve şifrelerinin nasıl geri alınabileceği ve bunun güvenli bir şekilde nasıl yapılacağı incelenirken kod inceleyicinin neleri göz önünde bulundurması gerektiği üzerinde durulacaktır.

### Genel hususlar
Kullanıcıyı (telefon sms'i, e-posta) ile bilgilendirin, öyle ki kullanıcı e-postada sitenize yönlendiren bir bağlantıya tıklamalı ve kullanıcıdan yeni bir şifre girmesini istemelidir.

Kullanıcının şifre değiştirmesine izin vermeden önce kullanıcıyı doğrulamak için kullanıcıdan zaten sahip olduğu oturum açma kimlik bilgilerini (Facebook, Twitter, Google, Microsoft Live, OpenID vb.) girmesini isteyin.

Kayıt veya şifremi unuttum kullanımını onaylamak için kullanıcıya bildirim gönderin.

Kayıtlı e-posta için hesap bilgilerinin değiştirildiğine dair bildirimler gönderin. Uygun zaman aşımı değerini ayarlayın. Örneğin, kullanıcı 48 saat içinde e-postaya yanıt vermezse, kullanıcı şifre değişikliğini yeniden onaylayana kadar sistemden dondurulacaktır.

• Kimlik ve paylaşılan gizli/şifre, veri gizliliği sağlamak için şifreleme kullanılarak aktarılmalıdır.

• Paylaşılan bir sır, bir mesaj kuyruğunda kısa bir süre için bile olsa asla açık metin biçiminde saklanamaz.
• Paylaşılan bir sır her zaman bir veritabanında karma veya şifrelenmiş biçimde saklanmalıdır.

• Şifrelenmiş paylaşılan sırrı depolayan kuruluşun kullanıcı parolalarını görüntüleme veya şifresini çözme yeteneğine ihtiyacı yoktur. Kullanıcı parolası asla bir kullanıcıya geri gönderilmemelidir.

• İstemcinin kullanıcı adı ve parolayı bir Web hizmetine yapılacak sonraki çağrılarda sunulmak üzere önbelleğe alması gerekiyorsa, kullanıcı adı ve parolayı korumak için güvenli bir önbellek mekanizmasının bulunması gerekir.

• Geçersiz bir girişi kullanıcıya geri bildirirken, kullanıcı adı ve/veya parolanın geçersiz olduğu belirtilmemelidir. Kullanıcı geri bildirim/hata mesajı hem kullanıcı adını hem de şifreyi tek bir "kullanıcı kimlik bilgisi" olarak değerlendirmelidir. Yani "Girdiğiniz kullanıcı adı veya şifre yanlış."

## 8.5 CAPTCHA

**Genel Bakış**

CAPTCHA ("Completely Automated Public Turing test to tell Computers and Humans Apart" ifadesinin kısaltmasıdır) bir erişim kontrol tekniğidir.

CAPTCHA, otomatik yazılımların Gmail, Hotmail ve Yahoo gibi web posta hizmetlerine erişerek e-posta spam'i oluşturmasını, tanıtım (ticari ve/veya siyasi) veya taciz ve vandalizm amacıyla bloglara, forumlara ve wiki'lere otomatik gönderiler yapmasını ve otomatik hesap oluşturmasını önlemek için kullanılır.

CAPTCHA'ların yararlı olduğu kanıtlanmış ve kullanımları mahkemede onaylanmıştır. CAPTCHA'nın atlatılması, ABD Mahkemelerinde Dijital Milenyum Telif Hakkı Yasası atlatma karşıtı bölüm 1201(a)(3) ve Avrupa Direktifi 2001/29/EC'nin ihlali olarak onaylanmıştır.

**Genel hususlar**

CAPTCHA'ların kod incelemesi, CAPT- CHA'nın güçlü güvenlik ilkeleriyle oluşturulduğundan emin olmak için incelemeyi yapan kişinin aşağıdaki kurallara dikkat etmesi gerekir.
• Yanlış bir denemeden sonra kullanıcının birden fazla tahmin girmesine izin vermeyin.

• Yazılım tasarımcısının ve kod incelemesinin tahmin etme statiğini anlaması gerekir. Yani Bir CAPTCHA tasarım dört (3 kedi ve 1 tekne) resim gösterir, Kullanıcıdan diğer resimlerle aynı kedi kategorisinde olmayan resmi seçmesi istenir. Otomatik yazılım her zaman ilk resmi seçerek %25'lik bir başarı oranına sahip olacaktır. İkincisi, CAPTCHA resimlerinin sabit havuzuna bağlı olarak, zaman içinde bir saldırgan doğru cevaplardan oluşan bir veritabanı oluşturabilir ve ardından %100 erişim elde edebilir.

• Cevabı HMAC (Karma tabanlı mesaj kimlik doğrulama kodu) kullanan sunucuya iletilen bir anahtar kullanmayı düşünün.

**Metin tabanlı CAPTCHA'lar aşağıdaki güvenlik tasarım ilkelerine uymalıdır...**

**1.** CAPTCHA uzunluğunu rastgele ayarlayın: Sabit bir uzunluk kullanmayın; saldırgana çok fazla bilgi verir.

**2.** Karakter boyutunu rastgele ayarlayın: Çeşitli yazı tipi boyutları / çeşitli yazı tipleri kullanarak saldırganın eğitimli tahminler yapamayacağından emin olun.

**3.** CAPTCHA'yı sallayın: CAPTCHA'yı sallamak saldırgan için zorluğu artırır.

**4.** Karmaşık bir karakter kümesi kullanmayın: Büyük bir karakter kümesi kullanmak CAPTCHA şemasının başarısını önemli ölçüde artırmaz.

güvenlik ve insan doğruluğuna gerçekten zarar verir.
**5.** CAPTCHA güvenliğini güçlendirmenin bir yolu olarak tanıma önleyici teknikler kullanın: Bazı karakterlerin döndürülmesi, ölçeklendirilmesi ve çeşitli yazı tipi boyutlarının kullanılması, karakter genişliğini daha az tahmin edilebilir hale getirerek tanıma verimliliğini azaltacak ve güvenliği artıracaktır.

**6.** Çizgiyi CAPTCHA'ların içinde tutun: Çizgiler CAPTCHA harflerinin sadece bazılarını geçmelidir, böylece bunun bir çizgi mi yoksa bir karakter parçası mı olduğu anlaşılamaz.

**7.** Büyük çizgiler kullanın: Karakter segmentleri kadar geniş olmayan satırların kullanılması saldırgana güçlü bir ayırt edici sağlar ve satır anti-segmentasyon tekniğini birçok saldırı tekniğine karşı savunmasız hale getirir.

**8.** CAPTCHA, ADA (1990 Engelli Amerikalılar Yasası) uyumlu olması gereken web siteleri için sorun yaratır. Kod gözden geçiricinin, web sitesinin yasalar gereği ADA şikayeti olması gereken yerlerde CAPTCHA uygulamasını gözden geçirmek için web erişilebilirlikleri ve güvenliği hakkında bilgi sahibi olması gerekebilir.

### Referanslar


- AMERİKA BİRLEŞİK DEVLETLERİ vs KENNETH LOWSON, KRISTOFER KIRSCH, LOEL STEVENSON Federal İddianame. 23 Şubat 2010. Erişim tarihi: 2012-01-02.
- http://www.google.com/recaptcha/captcha
- http://www.ada.gov/anprm2010/web%20anprm_2010.htm
- CAPTCHA'nın Erişilemezliği - Web'de Görsel Turing Testlerine Alternatifler http://www.w3.org/TR/turingtest/


## 8.6 Bant Dışı İletişime Genel

**Bakış**

'Bant dışı' terimi genellikle bir web uygulamasının son kullanıcı ile kullanıcının web tarayıcısı üzerinden gerçekleştirilen HTTP istek/cevaplarından ayrı bir kanal üzerinden iletişim kurması durumunda kullanılır. Yaygın örnekler arasında metin/SMS, telefon görüşmeleri, e-posta ve normal posta yer alır.

**Açıklama**

Bir uygulamanın son kullanıcı ile bu ayrı kanallar üzerinden iletişim kurmak istemesinin ana nedeni güvenliktir. Bir kullanıcı adı ve parola kombinasyonu, bir kullanıcının bir web sitesinin hassas olmayan bölümlerine göz atmasına ve bunları kullanmasına izin vermek için yeterli kimlik doğrulama olabilir, ancak daha hassas (veya riskli) işlevler daha güçlü bir kimlik doğrulama biçimi gerektirebilir. Bir kullanıcı adı ve parola virüslü bir bilgisayar, sosyal mühendislik, veritabanı sızıntısı veya diğer saldırılar yoluyla çalınmış olabilir, bu da web uygulamasının geçerli kullanıcı adı ve parola kombinasyonunun gerçekten amaçlanan kullanıcı olmasını sağlayan bir web oturumuna çok fazla güvenemeyeceği anlamına gelir.

**Hassas operasyonlara örnek olarak şunlar verilebilir:**

- Şifre değiştirme
- E-posta adresi, ev adresi vb. gibi hesap bilgilerinin değiştirilmesi
- Bir bankacılık uygulamasında fon aktarma
- Siparişlerin gönderilmesi, değiştirilmesi veya iptal edilmesi

Bu durumlarda birçok uygulama sizinle tarama oturumu dışında bir kanal üzerinden iletişim kuracaktır. Birçok büyük çevrimiçi mağaza, hesap bilgilerinizi değiştirdiğinizde veya ürün satın aldığınızda size onay e-postaları gönderir. Bu, bir saldırganın kullanıcı adı ve şifresine sahip olması durumunda, bir şey satın alırlarsa, meşru kullanıcı bir e-posta alır ve siparişi iptal etme veya web sitesini hesabı değiştirmedikleri konusunda uyarma şansına sahip olur.

Kimlik doğrulama için bant dışı teknikler kullanıldığında bu iki faktörlü kimlik doğrulama olarak adlandırılır.
Kimlik doğrulamanın üç yolu vardır:
**1.** Bildiğiniz bir şey (örn. şifre, parola, ezberlenmiş PIN)
**2.** Sahip olduğunuz bir şey (örn. cep telefonu, nakit kartı, RSA belirteci)
**3.** Olduğunuz bir şey (örn. iris taraması, parmak izi)

Bir bankacılık web sitesi kullanıcıların çevrimiçi işlem başlatmasına izin veriyorsa, 1) oturum açmak için kullanılan şifreyi alarak ve 2) kullanıcının kayıtlı telefonuna SMS yoluyla bir PIN numarası göndererek ve ardından kullanıcının işlemi tamamlamadan önce PIN'i girmesini isteyerek iki faktörlü kimlik doğrulama kullanabilir. Bu, kullanıcının bildiği (şifre) ve sahip olduğu (PIN'i almak için telefon) bir şey gerektirir.

Bir 'çip ve pin' banka kartı, kullanıcıların kartı yanlarında bulundurmalarını (sahip oldukları bir şey) ve ayrıca bir satın alma işlemi gerçekleştirirken bir PIN girmelerini (bildikleri bir şey) gerektirerek iki faktörlü kimlik doğrulama kullanacaktır. Bir 'çip ve pin' kartı PIN numarası dahilinde kullanılmaz, aynı şekilde kartınız yoksa PIN numarasını bilmek de işe yaramaz.

**Ne İncelenmeli**
Bant dışı işlevleri yerine getiren kod modüllerini incelerken, dikkat edilmesi gereken bazı yaygın sorunlar şunlardır:
**1.** Sistemin kötüye kullanılma riskinin farkında olun. Saldırganlar sitenizden SMS mesajları veya rastgele kişilere e-postalar göndermek isteyebilir. Emin olun:

**2.** Mümkün olduğunda, bant dışı bir özelliğin çağrılmasına neden olan bağlantılara yalnızca kimliği doğrulanmış kullanıcılar erişebilir (şifremi unuttum bir istisnadır).

**3.** Arayüzün hızını sınırlandırın, böylece virüslü makinelere veya saldırıya uğramış hesaplara sahip kullanıcılar bunu bir kullanıcıya bant dışı mesajlar göndermek için kullanamaz.

**4.** Özelliğin kullanıcıdan gelen hedefi kabul etmesine izin vermeyin, yalnızca kayıtlı telefon numaralarını, e-postaları, adresleri kullanın.

**5.** Yüksek riskli siteler için (örneğin bankacılık) kullanıcıların telefon numarası web sitesi yerine şahsen kaydedilebilir.

**6.** Bant dışı iletişimde herhangi bir kişisel bilgi veya kimlik doğrulama bilgisi göndermeyin.

**7.** Bant dışı kanallar üzerinden gönderilen PIN'lerin veya parolaların kısa ömürlü ve rastgele olduğundan emin olun.

**8.** SMS mesajlarının, o anda taramayı yürüten cihaza gönderilmesini önlemek de düşünülebilir.
tarama oturumu (yani SMS'in gönderildiği iPhone'unda gezinen kullanıcı). Ancak bunu uygulamak zor olabilir.
Mümkünse kullanıcılara kullanmak istedikleri bandı seçme şansı verin. Bankacılık siteleri için mobilde Zitmo kötü amaçlı yazılım cihazlar (bkz. referanslar) SMS mesajlarını engelleyebilir, ancak iOS cihazları henüz bu kötü amaçlı yazılımdan etkilenmemiştir, bu nedenle kullanıcılar SMS PIN'lerinin Apple cihazlarına gönderilmesini seçebilir ve Android'deyken PIN'i almak için kaydedilmiş sesli mesajları kullanabilirler.

**9.** Tipik bir dağıtımda, web uygulamasından ayrı özel donanım/yazılımlar
Geçici PIN'lerin ve muhtemelen şifrelerin oluşturulması da dahil olmak üzere bant dışı iletişim. Bu senaryoda PIN/parolayı web uygulamanıza göstermenize gerek yoktur (maruz kalma riskini artırır), bunun yerine web uygulaması son kullanıcı tarafından sağlanan PIN/parola ile özel donanım/yazılımı sorgulamalı ve

olumlu veya olumsuz bir yanıt.
Bankacılık sektörü de dahil olmak üzere birçok sektör, belirli işlem türleri için iki faktörlü kimlik doğrulamanın kullanılmasını gerektiren düzenlemelere sahiptir. Diğer durumlarda iki faktörlü kimlik doğrulama, dolandırıcılıktan kaynaklanan maliyetleri azaltabilir ve müşterilere bir web sitesinin güvenliği konusunda yeniden güvence verebilir.

### Referanslar
- https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
- http://securelist.com/blog/virus-watch/57860/new-zitmo-for-android-and-blackberry/

## 8.7 Oturum Yönetimine

**Genel Bakış**

Bir web oturumu, aynı kullanıcıyla ilişkili bir dizi ağ HTTP isteği ve yanıt işlemidir. Oturum yönetimi veya durumu, birden fazla istek süresince her bir kullanıcı hakkında bilgi veya durum saklanmasını gerektiren web uygulamaları tarafından gereklidir. Bu nedenle oturumlar, erişim hakları ve yerelleştirme ayarları gibi, kullanıcının oturum süresince web uygulamasıyla gerçekleştirdiği her bir etkileşim için geçerli olacak değişkenler oluşturma olanağı sağlar.

**Açıklama**

Kod inceleme uzmanının, geliştiricilerin hangi oturum tekniklerini kullandığını ve potansiyel güvenlik riskleri yaratabilecek açıkları nasıl tespit edeceğini anlaması gerekir. Web uygulamaları, ilk kullanıcı talebinden sonra anonim kullanıcıları takip etmek için oturumlar oluşturabilir. Buna örnek olarak kullanıcının dil tercihini korumak verilebilir. Ek olarak, web uygulamaları kullanıcı kimlik doğrulaması yaptıktan sonra oturumları kullanacaktır. Bu, sonraki taleplerde kullanıcıyı tanımlayabilmenin yanı sıra güvenlik erişim kontrollerini uygulayabilmeyi, kullanıcının özel verilerine yetkili erişimi ve uygulamanın kullanılabilirliğini artırmayı sağlar. Bu nedenle, mevcut web uygulamaları hem kimlik doğrulama öncesi hem de sonrası oturum yetenekleri sağlayabilir.

Oturum kimliği veya belirteci, kullanıcı kimlik doğrulama bilgilerini (bir kullanıcı oturumu şeklinde) kullanıcı HTTP trafiğine ve web uygulaması tarafından uygulanan uygun erişim kontrollerine bağlar. Modern web uygulamalarındaki bu üç bileşenin (kimlik doğrulama, oturum yönetimi ve erişim kontrolü) karmaşıklığı, ayrıca uygulama ve bağlamanın web geliştiricisinin elinde olması (web geliştirme çerçevesi bu modüller arasında sıkı ilişkiler sağlamadığından), güvenli bir oturum yönetimi modülünün uygulanmasını çok zor hale getirir.

Oturum kimliğinin ifşa edilmesi, ele geçirilmesi, tahmin edilmesi, kaba kuvvet uygulanması ya da sabitlenmesi, saldırganın web uygulamasındaki bir kurban kullanıcının kimliğine tamamen bürünebildiği oturum ele geçirme (ya da sidejacking) saldırılarına yol açacaktır. Saldırganlar hedefli veya genel olmak üzere iki tür oturum ele geçirme saldırısı gerçekleştirebilir. Hedefli bir saldırıda, saldırganın amacı belirli (veya ayrıcalıklı) bir web uygulaması kurban kullanıcısının kimliğine bürünmektir. Genel saldırılarda, saldırganın amacı web uygulamasındaki herhangi bir geçerli veya meşru kullanıcının kimliğine bürünmektir (veya bu kullanıcı olarak erişim sağlamaktır).

Güvenli oturum kimliklerinin uygulanması amacıyla, tanımlayıcıların (kimlikler veya belirteçler) oluşturulması aşağıdaki özellikleri karşılamalıdır:
- Oturum kimliği tarafından kullanılan isim aşırı derecede açıklayıcı olmamalı veya kimliğin amacı ve anlamı hakkında gereksiz ayrıntılar sunmamalıdır.

- Web geliştirme çerçevesinin varsayılan oturum kimliği adının "id" gibi genel bir adla değiştirilmesi önerilir.

- Oturum kimliği uzunluğu en az 128 bit (16 bayt) olmalıdır (Oturum kimliği değeri en az 64 bit entropi sağlamalıdır).

• Oturum kimliği içeriği (veya değeri), bir saldırganın kimliğin içeriğini çözebildiği ve kullanıcının, oturumun veya web uygulamasının iç işleyişinin ayrıntılarını çıkarabildiği bilgi ifşa saldırılarını önlemek için anlamsız olmalıdır.

SHA2 (256 bit) gibi kriptografik hash fonksiyonları kullanılarak kriptografik olarak güçlü oturum kimlikleri oluşturulması tavsiye edilir.

**Ne İncelenmeli**
Uygulamanız kimlik doğrulama içerdiğinde çerezleri zorunlu kılın. Kodu gözden geçiren kişinin uygulama çerezlerinde hangi bilgilerin saklandığını anlaması gerekir. Çerez için SSL gerektiren çerezde hassas bilgilerin saklanıp saklanmadığını ele almak için risk yönetimi gereklidir

### .Net ASPX web.config

**Örnek 8.2**

```
<Kimlik doğrulama modu="Formlar">
 <forms loginUrl="member_login.aspx"
  cookieless="UseCookies"
  requireSSL="true"
  path="/MyApplication" />
</authentication>
```

### Java web.xml

**Örnek 8.3**

```
<session-config>
 <cookie-config>
  <secure>true</secure>
 </cookie-config>
</session-config>
```

### PHP.INI

**Örnek 8.4**

```
void session_set_cookie_params ( int $lifetime [, string $path [, string $domain [, bool $secure = true [, bool
$httponly = true ]]]] )
```

**Oturum Sona Erme**
Oturum işleme kodunu gözden geçirirken, gözden geçirenin web uygulaması tarafından hangi sona erme zaman aşımlarına ihtiyaç duyulduğunu veya varsayılan oturum zaman aşımının kullanılıp kullanılmadığını anlaması gerekir. Web tarafından yetersiz oturum zaman aşımı

uygulama, saldırganın geçerli bir oturum kimliğini yeniden kullanabilmesi ve ilişkili oturumu ele geçirebilmesi için hala aktif olması gerektiğinden, diğer oturum tabanlı saldırıların maruziyetini artırır. Güvenli kodlama için hedeflerimizden birinin uygulamamızın saldırı yüzeyini azaltmak olduğunu unutmayın.

**.Net ASPX**

**Örnek 8.5**

```
<sistem.web>
  <sessionState
    mode="InProc"
    cookieless="true"
    timeout="15" />
 </system.web>
```

ASPX geliştiricisi bir oturum için varsayılan zaman aşımını değiştirebilir. Web.config dosyasındaki bu kod, oturum zaman aşımını 15 dakika olarak ayarlar. Bir aspx oturumu için varsayılan zaman aşımı 30 dakikadır.

**Java**

**Örnek 8.6**

```
<session-config>
   <session-timeout>1</session-timeout>
</session-config>
```

**PHP**

Bir oturum zaman aşımı mekanizmasına sahip değildir. PHP geliştiricilerinin kendi özel oturum zaman aşımlarını oluşturmaları gerekecektir.

**Oturum Oturumu Kapatma/Bitirme.**

Web uygulamaları, güvenlik bilincine sahip kullanıcıların web uygulamasını kullanmayı bitirdiklerinde oturumlarını aktif olarak kapatmalarına olanak tanıyan mekanizmalar sağlamalıdır.

.Net ASPX Session.Abandon() yöntemi, bir Session nesnesinde depolanan tüm nesneleri yok eder ve kaynaklarını serbest bırakır. Abandon yöntemini açıkça çağırmazsanız, oturum zaman aşımına uğradığında sunucu bu nesneleri yok eder. Kullanıcı oturumu kapattığında kullanmalısınız. Session.Clear() Oturumdan tüm anahtarları ve değerleri kaldırır. Oturum kimliğini değiştirmez. Kullanıcının yeniden giriş yapmasını ve oturuma özgü tüm verileri sıfırlamasını istemiyorsanız bu komutu kullanın.

**Oturum Saldırıları**

Genel olarak üç tür oturum saldırısı mümkündür:

- Oturum Korsanlığı: Birinin oturum kimliğini çalmak ve bunu o kullanıcının kimliğine bürünmek için kullanmak.

- Oturum Sabitleme: Birinin oturum kimliğini önceden tanımlanmış bir değere ayarlamak ve bu bilinen değeri kullanarak onu taklit etmek

- Oturum Yükseltme: Bir oturumun önemi değiştirildiğinde, ancak kimliği değiştirilmediğinde.

### Oturum Korsanlığı

• Çoğunlukla XSS saldırıları yoluyla yapılır, çoğunlukla HTTP-Only oturum çerezleri ile önlenebilir (Javascript kodu bunlara erişim gerektirmedikçe).

• Javascript'in oturum çerezlerine erişime ihtiyaç duymaması genellikle iyi bir fikirdir, çünkü XSS'nin tüm çeşitlerini önlemek genellikle bir sistemi sağlamlaştırmanın en zor kısmıdır.

• Oturum kimlikleri URL'lere değil çerezlerin içine yerleştirilmelidir. URL bilgileri tarayıcının geçmişinde ve HTTP Yönlendiricilerinde saklanır ve saldırganlar tarafından erişilebilir.

• Çerezlere varsayılan olarak javascript'ten erişilebildiğinden ve XSS'nin tüm çeşitlerini önlemek genellikle bir sistemi güçlendirmenin en zor kısmı olduğundan, bu erişimi yasaklayan "HTTPOnly" adlı bir özellik vardır. Oturum çerezinde bu özellik ayarlanmış olmalıdır. Her neyse, istemciden bir oturum çerezine erişmeye gerek olmadığından, bu erişime bağlı olan istemci tarafı kodundan şüphelenmelisiniz.

• Coğrafi konum kontrolü basit ele geçirme senaryolarını tespit etmeye yardımcı olabilir. Gelişmiş korsanlar kurbanın aynı IP'sini (veya aralığını) kullanır.

• Aktif bir oturuma başka bir konumdan erişildiğinde uyarı verilmelidir.

• Aktif bir kullanıcı, başka bir yerde aktif bir oturumu olduğunda uyarılmalıdır (politika tek bir kullanıcı için birden fazla oturuma izin veriyorsa).

### Oturum Sabitleme

• Uygulama havuzda bulunmayan yeni bir oturum kimliği görürse, reddedilmeli ve yeni bir oturum kimliği ilan edilmelidir. Bu, sabitlenmeyi önlemek için tek yöntemdir.

• Tüm oturum kimlikleri uygulama tarafından oluşturulmalı ve daha sonra kontrol edilmek üzere bir havuzda saklanmalıdır. Uygulama, oturum oluşturma konusunda tek yetkilidir.

### Oturum Yükseltme

• Bir oturum yükseldiğinde (oturum açma, oturum kapatma, belirli yetkilendirme), yuvarlanmalıdır.

• Birçok uygulama ziyaretçiler için de oturumlar oluşturur (yalnızca kimliği doğrulanmış kullanıcılar için değil). Kullanıcılar oturum açtıktan sonra uygulamanın kendilerine güvenli bir şekilde davranmasını beklediğinden, yükselme sırasında oturumu kesinlikle açmalıdırlar.

• Bir aşağı-yükselme meydana geldiğinde, daha yüksek seviyeye ilişkin oturum bilgileri temizlenmelidir.

• Oturumlar yükseltildiklerinde yuvarlanmalıdır. Rolling, session-id'nin değiştirilmesi ve oturum bilgilerinin yeni id'ye aktarılması gerektiği anlamına gelir.

### Oturum Yönetimi için Sunucu Tarafı Savunmaları

**.NET ASPX**

Yeni oturum kimlikleri oluşturmak, oturum yuvarlama, sabitleme ve ele geçirmeyi önlemeye yardımcı olur.

**Örnek 8.7**

```
public class GuidSessionIDManager : SessionIDManager {
public override string CreateSessionID(HttpContext context){
return Guid.NewGuid().ToString();
  }
  public override bool Validate(string id) {
try{
     Guid testGuid = new Guid(id);
     if (id == testGuid.ToString())
     return true;
   }catch(Exception e) { throw e }
   return false;
  }
 }
```

**Java**

**Örnek 8.8**

```
request.getSession(false).invalidate();
//ve sonra getSession(true)
(getSession()) ile yeni bir oturum
oluşturun
```

**PHP.INI**

**Örnek 8.9**

```
session.use_trans_sid = 0
session.use_only_cookies
```

**Referanslar**

- https://wwww.owasp.org/index.php/SecureFlag


A3

# A3 SITELER ARASI KOMUT DOSYASI OLUŞTURMA (XSS)

## 9.1 Genel Bakış

**Siteler Arası Komut Dosyası Oluşturma (XSS) nedir?**

Siteler arası komut dosyası oluşturma (XSS) bir tür kodlama açığıdır. Genellikle web uygulamalarında bulunur. XSS, saldırganların diğer kullanıcılar tarafından görüntülenen web sayfalarına kötü amaçlı yazılım enjekte etmesini sağlar. XSS, saldırganların aynı kaynak ilkesi gibi erişim kontrollerini atlamasına izin verebilir. Bu, OWASP Top ile buna uygun olarak bulunan en yaygın güvenlik açıklarından biridir.
10. Symantec yıllık tehdit raporunda XSS'nin web sunucularında bulunan iki numaralı güvenlik açığı olduğunu tespit etmiştir. Bu güvenlik açığının ciddiyeti/risk derecesi, güvenlik açığı bulunan site tarafından işlenen verilerin hassasiyetine ve sitenin organizasyonu tarafından uygulanan herhangi bir güvenlik azaltımının niteliğine bağlı olarak bir sıkıntıdan büyük bir güvenlik riskine kadar değişebilir.

### Açıklama

Üç tür XSS vardır: Yansıyan XSS (Kalıcı Olmayan), Saklanan XSS (Kalıcı) ve DOM tabanlı XSS. Bu türlerin her biri sunucuya kötü amaçlı bir yük iletmek için farklı araçlara sahiptir. Önemli olan sonuçların aynı olmasıdır.

### Ne İncelenmeli

Siteler arası komut dosyası oluşturma güvenlik açıklarını tespit etmek ve bir web uygulamasından kaldırmak zordur

Siteler arası komut dosyası kusurlarını tespit etmek ve bir web uygulamasından kaldırmak zor olabilir. Kusurları aramak için en iyi uygulama, yoğun bir kod incelemesi yapmak ve bir HTTP isteği yoluyla kullanıcı girdisinin HTML çıktısına girebileceği tüm yerleri aramaktır.

**Kod denetçisinin yakından incelemesi gerekir.**

**1.** Bu güvenilmeyen veriler HTML veya JavaScript ile aynı HTTP yanıtlarında iletilmez.

**2.** Veriler sunucudan istemciye iletildiğinde, güvenilmeyen veriler düzgün bir şekilde kodlanmalı ve HTTP yanıtı olmalıdır. Sunucudan gelen verilerin güvenli olduğunu varsaymayın. En iyi uygulama her zaman verileri kontrol etmektir.

**3.** DOM'a eklendiğinde, güvenilmeyen veriler aşağıdaki API'lerden biri kullanılarak eklenmelidir:
**a.** Node.textContent
**b.** document.createTextNode
**c.** Element.setAttribute (yalnızca ikinci parametre)

Kodu gözden geçiren kişi ayrıca HTML etiketlerinin (<img src...>, <iframe...>, <bgsound src...> vb. gibi) kötü amaçlı JavaScript iletmek için kullanılabileceğinin farkında olmalıdır.

Web uygulaması güvenlik açığı otomatik araçları/tarayıcıları Siteler Arası komut dosyası kusurlarını bulmaya yardımcı olabilir. Ancak tüm XSS açıklarını bulamazlar, bu nedenle manuel kod incelemeleri önemlidir. Manuel kod incelemeleri de her şeyi yakalayamayacaktır ancak risk seviyenize bağlı olarak derinlemesine savunma yaklaşımı her zaman en iyi yaklaşımdır.

OWASP Zed Attack Proxy (ZAP), web uygulamalarındaki güvenlik açıklarını bulmak için kullanımı kolay entegre bir sızma testi aracıdır. ZAP, otomatik tarayıcıların yanı sıra güvenlik açıklarını manuel olarak bulmanızı sağlayan bir dizi araç sağlar. Tarayıcınızı yönlendirdiğiniz bir web proxy'si gibi davranır, böylece bir siteye giden trafiği görebilir ve örümcek, tarama, fuzz ve uygulamaya saldırmanıza izin verir. Hem açık kaynak kodlu hem de ticari başka tarayıcılar

da mevcuttur.

### Microsft'un Anti-XSS kütüphanesini kullanın

XSS'yi önlemek için bir başka yardım seviyesi de bir Anti-XSS kütüphanesi kullanmaktır.

Ne yazık ki, HtmlEncode veya doğrulama özelliği, özellikle kullanıcı girdisinin JavaScript koduna, etiket özelliklerine, XML'e veya URL'ye eklenmesi gerekiyorsa, XSS ile başa çıkmak için yeterli değildir. Bu durumda iyi bir seçenek Anti-XSS kütüphanesidir

### .NET ASPX

**1.** ASPX .Net sayfalarında kod incelemesi, web yapılandırma dosyasının sayfa doğrulamayı kapatmadığından emin olmak için kontrol etmelidir. <pages validateRequest="false" />

**2.** .Net framework 4.0 sayfa doğrulamanın kapatılmasına izin vermez. Bu nedenle, programcı sayfa doğrulamayı kapatmak isterse, geliştiricinin 2.0 doğrulama moduna geri dönmesi gerekecektir. <httpRuntime requestValidation- Mode="2.0" />

**3.** Kodu gözden geçiren kişinin sayfa doğrulamanın hiçbir yerde kapatılmadığından emin olması ve kapatılmışsa nedenini ve kurumu hangi risklere açık hale getirdiğini anlaması gerekir. <%@ Page Language="C#" ValidationRequest="false"

### .NET MVC

MVC web uygulamaları kötü amaçlı XSS koduna maruz kaldığında, aşağıdaki gibi bir hata verirler:

**Şekil 6:** Örnek .Net XSS Çerçeve Hatası


**Bu güvenlik açığından kaçınmak için aşağıdaki kodun dahil edildiğinden emin olun:**
**<%server.HtmlEncode(stringValue)%>**

HTMLEncode yöntemi, belirtilen bir dizeye HTML kodlaması uygular. Bu, Web uygulamanızda kullanmadan önce form verilerini ve diğer istemci istek verilerini kodlamak için hızlı bir yöntem olarak kullanışlıdır. Kodlama verileri, potansiyel olarak güvenli olmayan karakterleri HTML kodlu eşdeğerlerine dönüştürür (MSDN,2013)

### JavaScript ve JavaScript Çerçeveleri

Hem Javascript hem de Javascript frameworkleri günümüzde web uygulamalarında yaygın olarak kullanılmaktadır. Bu durum, kodu gözden geçiren kişinin hangi frameworklerin XSS kusurlarını önlemede iyi iş çıkardığını ve hangilerinin çıkarmadığını bilmesini engellemektedir. Kodu gözden geçiren kişi, kullanılan çerçeve için herhangi bir CVE olup olmadığını kontrol etmeli ve ayrıca javascript çerçevesinin en son kararlı

sürüm olup olmadığını kontrol etmelidir.

**OWASP Referansları**
- OWASP XSS Önleme Hile Sayfası
- OWASP XSS Filtresi Kaçırma Hile Sayfası
- OWASP DOM tabanlı XSS Önleme Hile Sayfası
- Test Kılavuzu: Veri Doğrulama Testi ile ilgili 1. 3 bölüm
- OWASP Zed Saldırı Proxy Projesi

**Dış Referanslar**
- **https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdf**

- **https://cwe.mitre.org/data/definitions/79.html**

- **http://webblaze.cs.berkeley.edu/papers/scriptgard.pdf**

- **http://html5sec.org**

- **https://cve.mitre.org**

**9.2 HTML Öznitelik Kodlaması**
HTML öznitelikleri güvenilmeyen veriler içerebilir. Belirli bir sayfadaki HTML özniteliklerinden herhangi birinin güven sınırının dışından veri içerip içermediğini belirlemek önemlidir.

align, alink, alt, bgcolor, border, cellpadding, cellspacing, class, color, cols, colspan, coords, dir, face, height, hspace, ismap, lang gibi bazı HTML nitelikleri diğerlerinden daha güvenli kabul edilir, marginheight, marginwidth, multiple, nohref, noresize, noshade, nowrap, ref, rel, rev, rows, rowspan, scrolling, shape, span, summary, tabin- dex, title, usemap, valign, value, vlink, vspace, width.

XSS için kodu incelerken aşağıdaki gibi HTML özniteliklerine bakmamız gerekir XSS için kodu incelerken aşağıdaki gibi HTML özniteliklerine bakmamız gerekir:
**<input type="text" name="fname" value="GÜVENİLMEYEN VERİLER">**

Saldırılar aşağıdaki biçimde olabilir:
**"><script>/* kötü şeyler */</script>**

**Öznitelik kodlaması nedir?**
HTML öznitelik kodlaması, bir karakter dizisinin bir HTML öğesinin özniteliğini bozmasını önlemek için önemli olan bir karakter alt kümesinin yerini alır.

Bunun nedeni, niteliklerin doğası, içerdikleri veriler ve bir tarayıcı veya HTML ayrıştırıcı tarafından nasıl ayrıştırıldıkları ve yorumlandıklarının, bir HTML belgesinin ve öğelerinin nasıl okunduğundan farklı olmasıdır; OWASP XSS
Önleme Hile Sayfası. Alfanümerik karakterler dışında, özniteliğin değiştirilmesini önlemek için ASCII değeri 256'dan küçük olan tüm karakterleri &#xHH; biçimiyle (veya varsa adlandırılmış bir varlıkla) kaçışın. Bu kuralın bu kadar geniş olmasının nedeni, geliştiricilerin sıklıkla öznitelikleri tırnak içine almadan bırakmasıdır. Uygun şekilde tırnak içine alınmış özniteliklerden yalnızca ilgili tırnak işaretiyle kaçılabilir. Tırnak içine alınmamış nitelikler, [boşluk] % * + , - / ; < = > ^ ve | dahil olmak üzere birçok karakterle ayrılabilir.

Öznitelik kodlaması çeşitli şekillerde gerçekleştirilebilir. İki kaynak vardır:
**1.** HttpUtility.HtmlAttributeEncode

http://msdn.microsoft.com/en-us/library/wdek0zbf.aspx

**2.** OWASP Java Encoder Projesi
https://www.owasp.org/index.php/OWASP_Java_Encoder_Project

### HTML Varlığı

Kullanıcı tarafından kontrol edilen verileri veya güvenilmeyen kaynaklardan gelen verileri içeren HTML öğeleri, bağlamsal çıktı kodlaması için gözden geçirilmelidir. HTML varlıkları söz konusu olduğunda, HTML Varlık kodlamasının gerçekleştirilmesini sağlamaya yardımcı olmamız gerekir:

Güvenilmeyen veriler içeren örnek HTML Varlığı:

```
HTML Gövde Bağlamı   <span>GÜVENİLMEYEN VERİLER</span> VEYA < b o d y > . . . GÜVENİLMEYEN
VERİLER </body> VEYA
<div>GÜVENİLİR VERİLER </div>
```

HTML Varlık Kodlaması gereklidir

```
& --> &amp; < --> &lt; > --> &gt; " --> &quot; ' --> &#x27;
```

Güvenilmeyen verilerin varlık nesneleri içinde nereye/nereye yerleştirildiğinin gözden geçirilmesi önerilir. Aşağıdaki kodlayıcılar için kaynak kodunun araştırılması, HTML varlık kodlamasının uygulamada ve tutarlı bir şekilde yapılıp yapılmadığının belirlenmesine yardımcı olabilir.

### OWASP Java Kodlayıcı Projesi

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project

```
<input type="text" name="data" value="<%=Encode.forHtmlAttribute(dataValue) %>" />
```

### OWASP ESAPI

http://code.google.com/p/owasp-esapi-java/source/browse/trunk/src/main/java/org/owasp/esapi/codecs/HTMLEntityCodec.java

```
String safe = ESAPI.encoder().encodeForHTML( request.getParameter( "input" ) );
```

### JavaScript Parametreleri

Güvenilmeyen veriler, bir JavaScript işlevi/kodu içine yerleştiriliyorsa doğrulama gerektirir. Geçersiz kılınmış veriler veri bağlamından çıkabilir ve kullanıcı tarayıcısında kod bağlamında yürütülebilir.

İncelenmeye değer sömürü noktalarına (bataklara) örnekler:

```
<script>var currentValue='UNTRUSTED DATA';</script> <script>someFunction('UNTRUSTED DATA');</
script>attack                                                    :');/* BAD STUFF */
```

**Potansiyel çözümler:**
OWASP HTML Sanitizer Projesi
OWASP JSON Sanitizer Projesi

ESAPI JavaScript kaçışları bu şekilde çağrılabilir:

```
String safe = ESAPI.encoder().encodeForJavaScript( request.getParameter("input") );
```

Lütfen bazı JavaScript işlevlerinin güvenilmeyen verileri girdi olarak asla güvenli bir şekilde kullanamayacağını unutmayın - javascript kaçmış olsa bile!

Örneğin (bunun JavaScript'in nasıl kullanılmayacağına dair bir örnek olduğunu unutmayın):

```
<script> window.setInterval('...GÜVENİLMEYEN VERİLERDEN KAÇSANIZ BİLE BURADA XSS'LENİRSİNİZ...'); </script>
```

### eval()

**Örnek 9.1**

```
var txtField ="A1"; var txtUserInput ='"test@google.ie';alert(1);"; eval( "document.forms[0]." + txtField +
".value =" + A1);
```

### jquery

**Örnek 9.2**

```
var txtAlertMsg ="Merhaba Dünya:"; var txtUserInput ="test<script>alert(1)<\/script>"; $("#message").html(
txtAlertMsg +"" + txtUserInput +"");
```

Güvenli kullanım (metin kullanın, html değil)

```
$("#userInput")
```

**.text**

```
( "test<script>alert(1)<\/script>");<-- kullanıcı girdisini metin olarak değerlendirin
```

**İç İçe Bağlamlar**

Bu tür iç içe geçmiş bağlamlardan kaçınmak en iyisidir: bir JavaScript işlevini çağıran bir öğe niteliği vb. bu bağlamlar gerçekten zihninizi karıştırabilir.

```
<div onclick="showError('<%=request.getParameter("errorxyz")%>')">Bir hata oluştu </div>
Burada bir HTML özniteliğimiz (onClick) ve iç içe geçmiş bir Javascript fonksiyon çağrımız
(showError) var.
```

Tarayıcı bunu işlediğinde, önce onclick niteliğinin içeriğini HTML olarak çözecektir. Sonuçları JavaScript Yorumlayıcısına aktaracaktır. Yani burada 2 bağlamımız var... HTML ve Javascript (2 tarayıcı ayrıştırıcısı). "Katmanlı" kodlamayı DOĞRU sırada uygulamamız gerekir:

1. JavaScript kodlama
2. HTML Attribute Encode böylece düzgün bir şekilde "çözülür" ve güvenlik açığı oluşmaz.

```
<div onclick="showError ('<%= Encoder.encodeForHtml(Encoder.encodeForJavaScript( request.
getParameter("error")%>'))" >Bir hata oluştu</div>
```


A4

# A4 GÜVENSIZ DOĞRUDAN NESNE REFERANSI

## 10.1 Genel Bakış

Güvensiz Doğrudan Nesne Referansı, çeşitli düzeylerde erişim sağlayan veya dahili bir nesneyi kullanıcıya ifşa eden web uygulamalarında yaygın olarak görülen bir güvenlik açığıdır. İfşa edilebilecek şeylere örnek olarak veritabanı yeniden kabloları, URL'ler, dosyalar, hesap numaraları veya kullanıcıların URL'leri manipüle ederek web güvenlik kontrollerini atlamasına izin vermek verilebilir.

Kullanıcı web uygulamasına erişim yetkisine sahip olabilir, ancak bir veritabanı kaydı, belirli bir dosya veya hatta bir URL gibi belirli bir nesneye erişim yetkisine sahip olmayabilir. Potansiyel tehditler, web uygulamasının yetkili bir kullanıcısının, kullanıcının erişim yetkisi olmayan bir nesneye doğrudan işaret eden bir parametre değeri girmesinden kaynaklanabilir. Uygulama söz konusu nesne için kullanıcıyı doğrulamazsa, doğrudan nesne referansı hatası güvensiz bir duruma yol açabilir.

## 10.2 Açıklama

Bu risk sorununun kaynağı, sunucu tarafında önceden oluşturulmuş verilerin manipüle edilmesi veya güncellenmesine dayanmaktadır.

**SQL**
**Enjeksiyonunu**
**İncelemek İçin**
**Ne Yapmalı**

Bu güvenlik açığından yararlanan bir saldırı örneği, kullanıcının zaten doğrulanmış olduğu bir web uygulaması olabilir. Şimdi kullanıcı açık faturalarını başka bir web sayfası üzerinden görüntülemek istiyor. Uygulama, URL dizesini kullanarak hesap numarasını iletir. Uygulama, hesap bilgilerine erişen bir SQL çağrısında doğrulanmamış verileri kullanır:

**Örnek 10.1**

```
String query ="SELECT * FROM accts WHERE account = ?"; PreparedStatement pstmt =
connection. prepareStatement(query , ... );
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

Saldırgan, istediği hesap numarasını göndermek için tarayıcılarındaki 'acct' parametresini değiştirir. Uygulama kullanıcı doğrulaması yapmazsa, saldırgan yalnızca hedeflenen müşterinin hesabı yerine herhangi bir kullanıcının hesabına erişebilir.

**HTTP POST istekleri**

Bir Siber Güvenlik Analisti (Ibrahim Raafat) Yahoo'da güvensiz bir doğrudan nesne referansı güvenlik açığı buldu! Canlı HTTP Başlıklarını kullanarak POST isteğindeki içeriği kontrol edebildi:

```
prop=addressbook&fid=367443&crumb=Q4.PSLBfBe.&cid=1236547890&cmd= delete_comment
```

Burada 'fid' parametresi konu kimliği ve 'cid' ilgili yorum kimliğidir. Test ederken, şunları değiştirmeyi buldu

fid ve cid parametre değerleri, aslında başka bir kullanıcı tarafından gönderilen diğer yorumları forumdan silmesine izin verir.

Daha sonra, gönderi silme mekanizmasını test etmek için aynı yöntemi kullandı ve benzer bir güvenlik açığı buldu. Bir gönderiyi silmek için normal bir HTTP Header POST isteği şöyledir:

```
POST cmd=delete_item&crumb=SbWqLz.LDP0
```

URL'ye fid (konu kimliği) değişkenini ekleyerek diğer kullanıcıların ilgili gönderilerini silebileceğini keşfetti:

```
POST cmd=delete_item&crumb=SbWqLz.LDP0&fid=xxxxxx
```

Daha sonraki analizlerden sonra, bir saldırganın HTTP POST taleplerindeki parametreleri değiştirerek Yahoo kullanıcıları tarafından girilen 1,5 milyon kaydı silebileceğini buldu

**Dolaylı Referans Haritaları**
Dahası, saldırganlar dahili adlandırma kurallarını öğrenebilir ve işlem işlevselliği için yöntem adlarını çıkarabilir. Örneğin, bir uygulama bir nesnenin detay bilgilerini almak için URL'lere sahipse: Saldırganlar nesne üzerinde değişiklik yapmak için aşağıdaki URL'leri kullanmaya çalışacaktır:

```
xyz.com/Customers/View/2148102445 veya xyz.com/Customers/ViewDetails.aspx?ID=2148102445
```

Ayrıca web uygulaması dizin yolunun veya nesne adının bir kısmını listeleyen bir nesne döndürüyorsa, bir saldırgan bunları değiştirebilir.

```
xyz.com/Customers/Update/2148102445 veya
xyz.com/Customers/Modify.aspx?ID=2148102445 Veya xyz.com/Customers/admin
```

**Veri Bağlama Tekniği**
Tasarım çerçevelerinin çoğunda (JSP/Structs, Spring) görülen bir diğer popüler özellik de HTTP GET istek parametrelerinin veya HTTP POST değişkenlerinin doğrudan ilgili iş/komut nesnesinin değişkenlerine bağlandığı veri bağlamadır. Burada bağlama, bu tür sınıfların örnek değişkenlerinin, adlarına bağlı olarak istek parametresi değerleriyle otomatik olarak başlatılması anlamına gelir. Aşağıda verilen örnek tasarımı göz önünde bulundurun; iş mantığı sınıfının iş nesnesi sınıfını iş nesnesini istek parametreleriyle bağladığını gözlemleyin.

Bu tasarımdaki kusur, iş nesnelerinin istek parametrelerine bağlı olmayan değişkenlere sahip olabilmesidir. Bu değişkenler fiyat, azami limit, rol vb. gibi statik değerlere sahip veya bazı sunucu tarafı işleme mantığına bağlı anahtar değişkenler olabilir. Bu tür senaryolardaki bir tehdit, bir saldırganın istekte ek parametreler sağlayabilmesi ve iş nesnesi sınıfının açıkta olmayan değişkenlerine değer bağlamaya çalışmasıdır.

**Güvenli Tasarım Önerisi**:
• Burada dikkat edilmesi gereken önemli bir nokta, iş/form/komut nesnelerinin yalnızca kullanıcı girdilerine bağlı olan örnek değişkenlere sahip olması gerektiğidir.
• Ek değişkenler mevcutsa, bunlar özelliğin iş kuralıyla ilgili olanlar gibi hayati değişkenler olmamalıdır.

• Her durumda, uygulama kullanıcıdan sadece istenen girdileri kabul etmeli ve geri kalanı reddedilmeli veya bağlanmadan bırakılmalıdır. Ve eğer varsa, açık olmayan değişkenlerin başlatılması bağlama mantığından sonra gerçekleşmelidir.

**İnceleme Kriterleri**
Uygulama tasarımını gözden geçirin ve bir veri bağlama mantığı içerip içermediğini kontrol edin. İçeriyorsa, istek parametrelerine bağlanan iş nesnelerinin / fasulyelerin statik değerlere sahip olması gereken açık olmayan değişkenlere sahip olup olmadığını kontrol edin. Bu tür değişkenler bağlama mantığından önce başlatılırsa, bu saldırı başarıyla çalışacaktır.

### Kod İnceleme Görevlisinin yapması gerekenler:
Kod inceleme uzmanının "incelenen kodda kullanıcı girdisinin nesnelere doğrudan referans vermek için kullanıldığı tüm konumların haritasını çıkarması" gerekir. Bu konumlar, kullanıcı girdisinin bir veritabanı satırına, bir dosyaya, uygulama sayfalarına vb. erişmek için kullanıldığı yerleri içerir. Gözden geçirenin, nesnelere başvurmak için kullanılan girdinin değiştirilmesinin kullanıcının görüntüleme yetkisi olmayan nesnelerin alınmasına neden olup olamayacağını anlaması gerekir.

Sunucu tarafındaki nesnelere erişmek için güvenilmeyen girdi kullanılıyorsa, verilerin sızdırılmamasını sağlamak için uygun yetkilendirme kontrolleri kullanılmalıdır. Güvenilmeyen girdinin sunucu tarafındaki kod tarafından doğru şekilde anlaşıldığından ve kullanıldığından emin olmak için uygun girdi doğrulaması da gerekecektir. Bu yetkilendirme ve girdi doğrulamasının sunucu tarafında gerçekleştirilmesi gerektiğini unutmayın; istemci tarafındaki kod saldırgan tarafından atlatılabilir.

### MVC .NET'te bağlama sorunları
**Diğer adıyla Aşırı İlan Verme Diğer adıyla Toplu Atamalar**
MVC çatısında toplu atamalar, HTTP form alanlarındaki bir istekle gelen verilerle modellerimizi güncellememizi sağlayan bir mekanizmadır. Güncellenmesi gereken veriler form alanlarından oluşan bir koleksiyonda geldiği için, bir kullanıcı bir istek gönderebilir ve formda olmayan ve geliştiricinin güncellenmesini istemediği modeldeki diğer alanları değiştirebilir.

Oluşturduğunuz modellere bağlı olarak, değiştirilmesini istemediğiniz hassas veriler olabilir. Kötü niyetli bir kullanıcı, görünüm aracılığıyla kullanıcıya gösterilmeyen bir modelin alanlarını değiştirdiğinde ve gizli model değerlerini değiştirmek için kötü niyetli kullanıcı ek model parametreleri eklediğinde güvenlik açığından yararlanılır.

**Örnek 10.2**

```
public class kullanıcı
{
  public int ID { get; set; } <- görünüm aracılığıyla
  gösterilir public string Name { get; set; } <-
  görünüm aracılığıyla gösterilir
  public bool isAdmin{ get; set; } <-görünümden gizlendi
}
```

### İlgili görünüm (HTML)

**Örnek 10.3**

```
ID: <%= Html.TextBox("ID") %> <br>
   Ad: <%= Html.TextBox("Ad") %> <br>
            <-- burada isAdmin yok!
```

Bu model için karşılık gelen HTML 2 alan içerir: Kimlik ve Ad
Bir saldırgan isAdmin parametresini forma ekler ve gönderirse yukarıdaki model nesnesini değiştirebilir. Yani kötü niyetli bir saldırgan isAdmin=true olarak değiştirebilir

**Tavsiyeler:**
**1** - Kullanıcının düzenlememesi gereken değerlere sahip olmayan bir model kullanın.
**2** - Bind yöntemini ve güncellenebilen beyaz liste özniteliklerini kullanın.
**3** - Belirli öznitelik güncellemelerini hariç tutmak için controller.UpdateModel yöntemini kullanın.

### Referanslar
- Güvensiz Dir Nesne Referansları üzerine OWASP Top 10-2007
- ESAPI Erişim Referans Haritası API'si
- ESAPI Erişim Kontrol API'si (Bkz. isAuthorizedForData(), isAuthorizedForFile(), isAuthorizedForFunction() )
- https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project
- https://cwe.mitre.org/data/definitions/639.html
- https://cwe.mitre.org/data/definitions/22.html


A5

# A5 GÜVENLIK YANLIŞ YAPILANDIRMASI

Birçok modern uygulama çerçeveler üzerinde geliştirilmektedir. Bu çerçeveler geliştiriciye daha az iş çıkarır, çünkü çerçeve "ev işlerinin" çoğunu yapar. Geliştirilen kod, çerçevenin işlevselliğini genişletecektir. Burada, belirli bir çerçevenin ve uygulamanın gerçekleştirildiği dilin bilinmesi büyük önem taşımaktadır. İşlemsel işlevlerin çoğu geliştiricinin kodunda görünmeyebilir ve "üst" sınıflarda ele alınabilir. Gözden geçiren kişi altta yatan çerçevenin farkında olmalı ve bu konuda bilgili olmalıdır.

Web uygulamaları tek başlarına çalışmazlar, tipik olarak bir uygulama sunucusu çerçevesi içinde konuşlandırılırlar, bir ağ içinde fiziksel bir ana bilgisayarda bir işletim sistemi içinde çalışırlar.

Güvenli işletim sistemi yapılandırması (sertleştirme olarak da adlandırılır) genellikle kod incelemesi kapsamında değildir. Daha fazla bilgi için İnternet Güvenliği Merkezi işletim sistemi kıyaslamalarına bakın.

Günümüzde ağlar, taşıma hizmetleri sağlayan yönlendiriciler ve anahtarlardan çok daha fazlasını içermektedir. Filtreleme anahtarları, VLAN'lar (sanal LAN'lar), güvenlik duvarları, WAF'lar (Web Uygulaması Güvenlik Duvarı) ve çeşitli ara kutular (örneğin ters proxy'ler, saldırı tespit ve önleme sistemleri), yapılandırıldıklarında kritik güvenlik hizmetleri sağlarlar. Bu büyük bir konudur, ancak bu web uygulaması kod inceleme kılavuzunun kapsamı dışındadır. İyi bir özet için SANS (System Administration, Networking, and Security) Institute Critical Control 10: Secure Configu- rations for Network Devices such as Firewalls, Routers, and Switches.

Uygulama sunucusu çerçeveleri güvenlikle ilgili birçok özelliğe sahiptir. Bu yetenekler, genellikle XML biçiminde statik yapılandırma dosyalarında etkinleştirilir ve yapılandırılır, ancak kod içinde ek açıklamalar olarak da ifade edilebilir.

### 11.1 Apache Struts

Struts'ta struts-config.xml ve web.xml dosyaları bir uygulamanın işlemsel işlevselliğini görüntülemek için temel noktalardır. struts-config.xml dosyası her HTTP isteği için eylem eşlemelerini içerirken, web.xml dosyası dağıtım tanımlayıcısını içerir.

Struts çerçevesi, girdi verilerini doğrulamak için düzenli ifadelere dayanan bir doğrulayıcı motora sahiptir. Doğrulayıcının güzelliği, her form bean için kod yazılmasına gerek olmamasıdır. (Form bean, HTTP isteğinden verileri alan Java nesnesidir). Doğrulayıcı struts'ta varsayılan olarak etkin değildir. Doğrulayıcıyı etkinleştirmek için struts-config.xml dosyasının <plug-in> bölümünde bir eklenti tanımlanmalıdır. Tanımlanan özellik, struts çerçevesine özel doğrulama kurallarının nerede tanımlandığını (validation.xml) ve gerçek kuralların kendilerinin bir tanımını (validation-rules.xml) söyler.

Struts çerçevesini doğru bir şekilde anlamadan ve sadece Java kodunu denetleyerek, herhangi bir doğrulamanın yürütüldüğünü göremezsiniz ve tanımlanan kurallar ile Java işlevleri arasındaki ilişkiyi göremezsiniz.

Eylem eşlemeleri, bir istek alındığında uygulama tarafından gerçekleştirilen eylemi tanımlar. Burada, **örnek 11.1'de,** URL "/login" içerdiğinde LoginAction'ın çağrılacağını görebiliriz. Eylem eşlemelerinden, harici girdi alındığında uygulamanın gerçekleştirdiği işlemleri görebiliriz.

**Örnek 11.1**

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
      "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
<struts-config>
<form-beans>
  <form-bean name="login" type="test.struts.LoginForm" />
</form-beans>
<global-forwards>
</global-forwards>
<eylem eşlemeleri>
  <action
     path="/login"
     type="test.struts.LoginAction" >
        <forward name="valid" path="/jsp/MainMenu.jsp"/>
        <forward name="invalid" path="/jsp/LoginView.jsp"/>
  </action>
</action-mappings>
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
    value="/test/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml"/>
</plug-in>
</struts-config>
```

### 11.2 Java Enterprise Edition Bildirimsel Yapılandırma

Bazı güvenlik özelliklerine bir Java programı içinden erişilebilir. Programatik güvenlik, çerçeveye özgü veya standart Java Enterprise Edition (JEE) çerçeve API'leri kullanılarak web uygulaması içinde yapılır. JEE, uygulama kaynaklarına erişimin güvenlik rolüne dayalı olarak verildiği rol tabanlı bir güvenlik modeli kullanan JEE güvenlik modeline sahiptir. Güvenlik rolü, sorumluların (kimliği doğrulanmış varlıklar, genellikle bir kullanıcı) mantıksal bir gruplamasıdır ve erişim, rol üzerinde bir güvenlik kısıtlaması belirtilerek beyan edilir.

Kısıtlamalar ve roller, XML öğeleri olarak ifade edilen dağıtım tanımlayıcıları olarak ifade edilir. Farklı bileşen türleri, dağıtım tanımlayıcıları için farklı biçimler veya şemalar kullanır:

• Web bileşenleri web.xml dosyasında bir web uygulaması dağıtım tanımlayıcısı kullanabilir

• Enterprise JavaBeans bileşenleri META-INF/ejb-jar.xml adlı bir EJB dağıtım tanımlayıcısı kullanabilir

Dağıtım tanımlayıcısı kaynakları (örneğin, belirli bir URL üzerinden erişilebilen servlet'ler), hangi rollerin kaynağa erişim yetkisine sahip olduğunu ve erişimin nasıl kısıtlandığını (örneğin, GET ile ancak POST ile değil) tanımlayabilir.

**Örnek 11.2**'deki örnek web bileşeni tanımlayıcısı ("web.xml" dosyasına dahil edilmiştir) bir Katalog servleti, bir "yönetici" rolü, servlet içinde GET ve POST istekleriyle erişilebilen bir SalesInfo kaynağı tanımlar ve yalnızca "yönetici" rolüne sahip, SSL kullanan ve HTTP temel kimlik doğrulamasını başarıyla kullanan kullanıcılara erişim izni verilmesi gerektiğini belirtir.

**Örnek 11.2**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd" version=?2.5?>

  <display-name>Güvenli Bir Uygulama</display-name>
  <servlet>
   <servlet-name>catalog</servlet-name>
   <servlet-class>com.mycorp.CatalogServlet</servlet-class>
   <init-param>
    <param-name>katalog</param-name>
    <param-value>Bahar</param-value>
   </init-param>

   <!-- Güvenlik Rollerini Tanımlayın –>
   <security-role-ref>
    <rol-adı>MGR</rol-adı>
    <role-link>yönetici</role-link>
   </security-role-ref>
  </servlet>

  <güvenlik-rolü>
   <rol-adı>yönetici</rol-adı>
  </security-role>
  <servlet-mapping>
   <servlet-name>catalog</servlet-name>
   <url-pattern>/catalog/*</url-pattern>
  </servlet-mapping>

  <!-- Bir Güvenlik Kısıtı Tanımlayın –>
  <güvenlik kısıtı>

  <!-- Korunacak Kaynakları Belirtin –>
  <web-resource-collection>
   <web-resource-name>SalesInfo</web-resource-name>
   <url-pattern>/salesinfo/*</url-pattern>
   <http-method>GET</http-method>
   <http-method>POST</http-method>
  </web-resource-collection>

  <!-- Korumalı Kaynaklara Hangi Kullanıcıların Erişebileceğini Belirtin –>
  <auth-constraint>
   <rol-adı>yönetici</rol-adı>
  </auth-constraint>
```

```
  <!-- SSL kullanarak Güvenli Aktarımı belirtin (gizli garanti) -->
   <kullanıcı-veri-kısıtı>
    <transport-guarantee>GİZLİ</transport-guarantee>
   </user-data-constraint>
   </security-constraint>

   <!-- HTTP Temel Kimlik Doğrulama Yöntemini Belirtin -->
   <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>dosya</realm-name>
   </login-config>
</web-app>
```

Güvenlik rolleri, şekil **örneği 11.3**'te görüldüğü gibi "ejb-jar.xml" dosyasında kurumsal Java fasulyeleri için de bildirilebilir.

**Örnek 11.3**

```
<ejb-jar>
   <assembly-descriptor>
      <güvenlik-rolü>
         <açıklama>Tek uygulama rolü</açıklama>
         <role-name>TheApplicationRole</role-name>
      </security-role>
   </assembly-descriptor>
</ejb-jar>
```

Ancak fasulyeler için, servlet'ler içindeki kaynaklara erişim belirtmek yerine, fasulye yöntemlerine erişim belirtilir. **Örnek 11.4**'teki örnekte, birkaç fasulye için çeşitli yöntem erişim kısıtlamaları gösterilmektedir.

**Örnek 11.4**

```
<ejb-jar>
   <assembly-descriptor>
      <method-permission>
         <açıklama>Çalışan ve geçici çalışan rolleri EmployeeService fasulyesinin
            herhangi bir yöntemine erişebilir </açıklama>
         <rol-adı>çalışan</rol-adı>
         <role-name>temp-employee</role-name>
         <metod>
            <ejb-name>EmployeeService</ejb-name>
            <method-name>*</method-name>
         </method>
      </method-permission>
```

```
<method-permission>
  <açıklama>Çalışan rolü, AardvarkPayroll fasulyesinin
    findByPrimaryKey, getEmployeeInfo ve updateEmployeeInfo(String)
    yöntemlerine erişebilir </açıklama>
  <rol-adı>çalışan</rol-adı>
  <metod>
    <ejb-name>AardvarkPayroll</ejb-name>
    <method-name>findByPrimaryKey</method-name>
  </method>
  <metod>
    <ejb-name>AardvarkPayroll</ejb-name>
    <method-name>getEmployeeInfo</method-name>
  </method>
  <metod>
    <ejb-name>AardvarkPayroll</ejb-name>
    <method-name>updateEmployeeInfo</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </method>
</method-permission>
<method-permission>
  <açıklama>Yönetici rolü EmployeeServiceAdmin fasulyesinin
    herhangi bir yöntemine erişebilir </açıklama>
  <rol adı>admin</rol adı>
  <metod>
    <ejb-name>EmployeeServiceAdmin</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
<method-permission>
  <açıklama>Kimliği doğrulanmış herhangi bir kullanıcı
    EmployeeServiceHelp fasulyesinin herhangi bir yöntemine
    erişebilir</açıklama>
  <işaretlenmemiş/>
  <metod>
    <ejb-name>EmployeeServiceHelp</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
<hariç tutma listesi>
  <açıklama>Bu dağıtımda EmployeeFiring bean'inin hiçbir fireTheCTO
    yöntemi kullanılamaz</açıklama>
  <metod>
    <ejb-name>EmployeeFiring</ejb-name>
    <method-name>fireTheCTO</method-name>
  </method>
</exclude-list>
```

```
    </assembly-descriptor>
</ejb-jar>
```

Uygulamanın güvenliğini sağlamak için XML dağıtım tanımlayıcıları kullanılıyorsa, erişim denetimlerinin doğru rollere düzgün şekilde uygulandığından ve kimlik doğrulama yöntemlerinin beklendiği gibi olduğundan emin olmak için kod incelemesi "web.xml" ve "ejb-jar.xml" dosyalarını içermelidir.

### 11.3 JEE Ek Açıklamaları

Güvenlik için JEE ek açıklamaları Java Dokümanlarında tanımlanmıştır [2]. Mevcut ek açıklamalar şunlardır:

- **@DeclareRoles**
- **@DenyAll -** hiçbir rol yöntemi çağıramaz.
- **@PermitAll -** tüm roller yöntemi çağırabilir.
- **@RolesAllowed -** yöntemi çağırmasına izin verilen roller.
- **@RunAs -** yöntemi dinamik olarak belirli bir rol olarak çalıştırır.

**Örneğin, örnek 11.5'teki** kod, çalışanların ve yöneticilerin kalıcı mağazaya film eklemesine, herkesin filmleri listelemesine, ancak yalnızca yöneticilerin filmleri silmesine izin verir.

**Örnek 11.5**

```
public class Filmler {

    private EntityManager entityManager;

    @RolesAllowed({"Employee", "Manager"})
    public void addMovie(Movie film) throws Exception {
        entityManager.persist(film);
    }

    @RolesAllowed({"Yönetici"})
    public void deleteMovie(Movie film) throws Exception {
        entityManager.remove(film);
    }

    @PermitAll
    public List<Movie> getMovies() throws Exception {
        Query query = entityManager.createQuery("SELECT m from Movie as m");
        return query.getResultList();
    }
}
```

Kod incelemesi bu tür ek açıklamaları aramalıdır. Varsa, doğru rolleri ve izinleri yansıttıklarından ve "ejb-jar.xml" dosyasında bildirilen rol izinleriyle tutarlı olduklarından emin olun.

## 11.4 Çerçeveye Özel Yapılandırma

### Apache Tomcat

Güvenlikle ilgili parametrelerin beklendiği gibi yapılandırıldığından emin olmak için server.xml[3] dosyası gözden geçirilmelidir. tom- cat server.xml dosyası güvenlikle ilgili birçok parametre tanımlar.

**Tablo 12:** Apache Tomcat Güvenlik Parametreleri

| Parametre | Açıklama |
|---|---|
| Sunucu | Bu, kapatma bağlantı noktasıdır. |
| Konektörler | maxPostSize, maxParameterCount, server, SSLEnabled, secure, ciphers. |
| Ev sahibi | autoDeploy, deployOnStartup, deployXML |
| Bağlam | crossContext, privileged, allowLinking |
| Filtre | Tomcat, gelen istekler için yapılandırılabilecek bir dizi filtre sağlar |

Filtreler özellikle güçlüdür ve kod incelemesi, zorlayıcı bir neden olmadığı sürece kullanıldıklarını doğrulamalıdır.

### Jetty

Jetty çeşitli güvenlik geliştirmeleri ekler:

- Form içeriğinin sınırlandırılması
- Şifreleri gizleme

Maksimum form içeriği boyutu ve form anahtarlarının sayısı "jetty-web.xml" dosyasında sunucu ve web uygulaması düzeyinde yapılandırılabilir.

**Örnek 11.6**

```
<Configure class="org.eclipse.jetty.webapp.WebAppContext">
 ...
 <Set name="maxFormContentSize">200000</Set>
 <Set name="maxFormKeys">200</Set>
</Configure>

<configure class="org.eclipse.jetty.server.Server">
 ...
 <Call name="setAttribute">
  <Arg>org.eclipse.jetty.server.Request.maxFormContentSize</Arg>
  <Arg>100000</Arg>
 </Call>
 <Call name="setAttribute">
  <Arg>org.eclipse.jetty.server.Request.maxFormKeys</Arg>
  <Arg>2000</Arg>
 </Call>
</configure>
```

Jetty, genellikle düz metin parolasının gerekli olduğu jetty XML dosyalarında gizlenmiş parolaların kullanımını da destekler. **örnek 11.7, gizlenmiş** bir JDBC Veri Kaynağı için parola belirleyen örnek kodu göstermektedir (gizlenmiş parola Jetty org.eclipse.jetty.util.security.Password yardımcı programı tarafından oluşturulur).

**Örnek 11.7**

```
<New id="DSTest" class="org.eclipse.jetty.plus.jndi.Resource">
<Arg></Arg>
<Arg>jdbc/DSTest</Arg>
<Arg>
        <New class="com.jolbox.bonecp.BoneCPDataSource">
                <Set name="driverClass">com.mysql.jdbc.Driver</Set>
                <Set name="jdbcUrl">jdbc:mysql://localhost:3306/foo</Set>
                <Set name="username">dbuser</Set>
                <Set name="password">
                <Call class="org.eclipse.jetty.util.security.Password" name="deobfuscate">
                        <Arg>OBF:1ri71v1r1v2n1ri71shq1ri71shs1ri71v1r1v2n1ri7</Arg>
                </Call>
                </Set>
                <Set name="minConnectionsPerPartition">5</Set>
                <Set name="maxConnectionsPerPartition">50</Set>
                <Set name="acquireIncrement">5</Set>
                <Set name="idleConnectionTestPeriod">30</Set>
        </Yeni>
</Arg>

</Yeni>
```

### JBoss AS

JBoss Uygulama Sunucusu, Jetty gibi, XML yapılandırma dosyalarında parola gizlemeye (JBoss'ta parola maskeleme olarak adlandırılır) izin verir. Parola maskesi oluşturmak için JBoss parola yardımcı programını kullandıktan sonra, XML yapılandırma dosyalarında maskelenmiş bir parolanın herhangi bir oluşumunu aşağıdaki ek açıklama ile değiştirin.

**Örnek 11.8**

```
<annotation>@org.jboss.security.integration.password.Password
            (securityDomain=MASK_NAME,methodName=setPROPERTY_NAME)
</annotation>
```

JBoss Güvenlik Kılavuzu'ndaki XML Yapılandırmasında Parolaları Maskeleme bölümüne bakın.

### Oracle WebLogic

WebLogic server, **tablo 13**'te gösterildiği gibi "weblogic.xml" dosyasında ek dağıtım tanımlayıcılarını destekler.

**Tablo 13:** Weblogic Güvenlik Parametreleri

| Parametre | Açıklama |
|---|---|
| harici olarak tanımlanmış | Rol - asıl eşlemeleri WebLogic Admin Console'da harici olarak tanımlanır |
| run-as-principal-name | Bir rol olarak çalışırken o role bir sorumlu atama |
| run-as-role-assignment | Run-as-principal-name tanımlayıcısını içerir |
| güvenlik-izni | Security-permission-spec tanımlayıcısını içerir |
| Filtre | Tomcat, gelen istekler için yapılandırılabilecek bir dizi filtre sağlar |
| security-permission-spec | Uygulama izinlerini belirtin [6] |
| security-role-assignment | Sorumluları açıkça bir role atama |

WebLogic ek dağıtım tanımlayıcıları hakkında daha fazla bilgi weblogic.xml Deployment Descriptors adresinde bulunabilir.

WebLogic içinde çalışan web uygulamalarının güvenliğini sağlamaya ilişkin genel yönergeler için WebLog- ic Güvenliği Programlama kılavuzuna ve NSA'nın BEA WebLogic Platform Güvenlik Kılavuzu'na bakın.

### 11.5 Programatik Yapılandırma: J2EE

Programatik güvenlik için J2EE API, EJBContext arayüzü ve HttpServle- tRequest arayüzünün yöntemlerinden oluşur. Bu yöntemler, bileşenlerin arayanın veya uzak kullanıcının güvenlik rolüne dayalı olarak iş mantığı kararları vermesine olanak tanır (kullanıcıların kimliklerini doğrulamak için de yöntemler vardır, ancak bu güvenli dağıtım yapılandırmasının kapsamı dışındadır).

**J2EE güvenlik yapılandırması ile etkileşime giren J2EE API'leri şunları içerir:**

- getRemoteUser, istemcinin kimlik doğrulamasını yaptığı kullanıcı adını belirler
- isUserInRole, uzaktaki bir kullanıcının belirli bir güvenlik rolünde olup olmadığını belirler.
- getUserPrincipal, geçerli kullanıcının asıl adını belirler ve bir java.security.

Yapılandırma ile tutarlılığı sağlamak için bu programatik API'lerin kullanımı gözden geçirilmelidir. Özellikle, security-role-ref öğesi, isUserInRole yöntemine aktarılacak rol adını içeren bir role-name alt öğesiyle birlikte dağıtım tanımlayıcısında bildirilmelidir.

**Örnek 11.9**'daki kod, programatik oturum açma ve kimlikler ile roller oluşturma amacıyla programatik güvenliğin kullanımını göstermektedir. Bu servlet aşağıdakileri yapar:

- geçerli kullanıcı hakkındaki bilgileri görüntüler.
- kullanıcıdan oturum açmasını ister.
- oturum açma yönteminin etkisini göstermek için bilgileri tekrar yazdırır.

• Kullanıcıyı dışarı çıkarır.

• oturum kapatma yönteminin etkisini göstermek için bilgileri tekrar yazdırır.

**Örnek 11.9**

```
enterprise.programmatic_login paketi;

import java.io.*;
import java.net.*;
import javax.annotation.security.DeclareRoles;
import javax.servlet.*;
import javax.servlet.http.*;

@DeclareRoles("javaee6user")
public class LoginServlet extends HttpServlet {

   /**
    * Hem HTTP GET hem de POST yöntemleri için istekleri işler.
    * @param request servlet isteği
    * @param response servlet yanıtı
    */
   protected void processRequest(HttpServletRequest request,
           HttpServletResponse response)
       throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      dene {
         String userName = request.getParameter("txtUserName");
         String password = request.getParameter("txtPassword");

         out.println("Oturum Açmadan Önce" + "<br><br>");
         out.println("IsUserInRole?.."
                 + request.isUserInRole("javaee6user")+"<br>");
         out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
         out.println("getUserPrincipal?.."
                 + request.getUserPrincipal()+"<br>");
         out.println("getAuthType?.." + request.getAuthType()+"<br><br>");

         dene {
            request.login(userName, password);
         } catch(ServletException ex) {
            out.println("Login Failed with a ServletException..."
               + ex.getMessage());
            return;
         }
         out.println("Oturum Açıldıktan Sonra... "+"<br><br>");
         out.println("IsUserInRole?.."
```

```
            + request.isUserInRole("javaee6user")+"<br>");
        out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
        out.println("getUserPrincipal?.."
            + request.getUserPrincipal()+"<br>");
        out.println("getAuthType?.." + request.getAuthType()+"<br><br>");

        request.logout();
        out.println("Oturum Kapatıldıktan Sonra... "+"<br><br>");
        out.println("IsUserInRole?.."
            + request.isUserInRole("javaee6user")+"<br>");
        out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
        out.println("getUserPrincipal?.."
            + request.getUserPrincipal()+"<br>");
        out.println("getAuthType?.." + request.getAuthType()+"<br>");
      } finally {
        out.close();
      }
    }
    ...
}
```

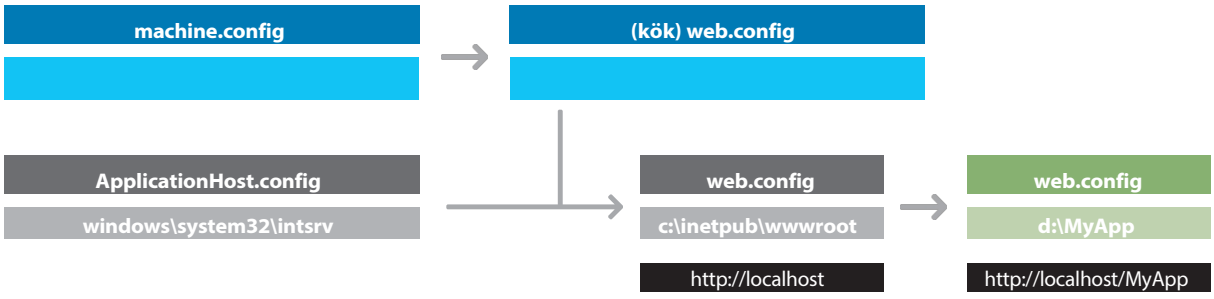
Daha ayrıntılı bilgi Java EE Tutorial'da bulunabilir: Web Uygulamaları ile Programatik Güvenliği Kullanma.

### 11.6 Microsoft IIS

ASP.NET / IIS uygulamaları, uygulama yapılandırma ayarlarını korumak için web.config adlı isteğe bağlı XML tabanlı bir yapılandırma dosyası kullanır. Bu, kimlik doğrulama, yetkilendirme, hata sayfaları, HTTP ayarları, hata ayıklama ayarları, web hizmeti ayarları vb. gibi konuları kapsar. Bu dosyalar hakkında bilgi sahibi olmadan işlem analizi yapmak çok zor ve doğru olmayacaktır.

IIS 7'de hiyerarşi seviyesini etkileyen bir yapılandırma sistemi vardır ve bir dosyanın

**Şekil 7:** IIS Yapılandırma Dosyaları




başka bir şey. Aşağıdaki şekilde bunun nasıl çalışacağı ve her bir dosyanın konumu gösterilmektedir (Aguilar, 2006)

Bir web uygulaması için sanal dizinin kökünde bir web.config dosyası sağlamak mümkündür. Dosya yoksa, machine.config dosyasındaki varsayılan yapılandırma ayarları kullanılır. Dosya mevcutsa, web.config dosyasındaki tüm ayarlar

**Örnek 11.10**

```
<Kimlik doğrulama modu="Formlar">
  <forms name="name"
    loginUrl="url"
    protection="Encryption"
    timeout="30" path="/"
    requireSSL="true|"
    slidingExpiration="false">
    <credentials passwordFormat="Clear">
      <kullanıcı adı="kullanıcı adı" parola="parola"/>
    </credentials>
  </forms>
  <passport redirectUrl="internal"/>
</authentication>
```

web.config varsayılan ayarları geçersiz kılacaktır.
Önemli güvenlik ayarlarının çoğu kodda değil, çerçeve yapılandırma dosyalarında ayarlanır. Çerçeve tabanlı uygulamaları incelerken çerçeve hakkında bilgi sahibi olmak büyük önem taşır.

**Tablo 14:** Web.config Dosyasındaki

| Parametre | Açıklama |
|---|---|
| kimlik doğrulama modu | Varsayılan kimlik doğrulama modu ASP.NET form tabanlı kimlik doğrulamadır. |
| loginUrl | Geçerli bir kimlik doğrulama çerezi bulunamazsa, isteğin oturum açma için yeniden yönlendirileceği URL'yi belirtir. |
| koruma | Çerezin 3DES veya DES kullanılarak şifrelendiğini ancak çerez üzerinde DV gerçekleştirilmediğini belirtir. Düz metin saldırılarına karşı dikkatli olun. |
| zaman aşımı | Dakika cinsinden çerez son kullanma süresi. |

web.config dosyasındaki çerçeveye özgü parametrelerin bazı örnekleri **tablo 14'te** gösterilmektedir.

## 11.7 Çerçeveye Özel Yapılandırma: Microsoft IIS

Güvenlik özellikleri IIS'de <system.webServer><security> bölümündeki Web.config (uygulama düzeyi) veya ApplicationHost.config (sunucu düzeyi) dosyası kullanılarak yapılandırılabilir. Yapılandırılabilecek özellik türleri şunları içerir:

- İzin verilen kimlik doğrulama yöntemleri
- Yetkilendirme kuralları
- Filtreler ve limitler talep edin
- SSL Kullanımı
- Kaynak IP adresi filtreleme
- Hata işleme

Web.config ve ApplicationHost.config dosyaları kod incelemesine dahil edilmelidir. <system.web-

Server><security> bölümleri, tüm güvenlik yapılandırmasının beklendiği gibi olduğundan emin olmak için gözden geçirilmelidir.

Microsoft IIS'nin genel yapılandırmasının güvenliğini sağlamaya ilişkin yönergeler için, IIS temel, istemci sertifikası, özet, IIS istemci sertifikası ve Windows kimlik doğrulama yöntemlerini destekler. Bunlar <system.web- Server><security><authentication> bölümünde yapılandırılır.

**Örnek 11.11**'deki örnek, MySite adlı bir site için anonim kimlik doğrulamayı devre dışı bırakır, ardından her ikisini de etkinleştirir

**Örnek 11.11**

```
<location path="MySite">
  <system.webServer>
    <güvenlik>
      <kimlik doğrulama>
        <anonymousAuthentication enabled="false" />
        <basicAuthentication enabled="true" defaultLogonDomain="MySite" />
        <windowsAuthentication enabled="true" />
       </authentication>
    </güvenlik>
  </system.webServer>
</location>
```

site için temel kimlik doğrulama ve windows kimlik doğrulama.
IIS yetkilendirme yapılandırması, kullanıcıların siteye veya sunucuya erişiminin belirlenmesine olanak tanır ve **<system.webServer><security><authorization>** bölümü.

**Örnek 11.12'**deki yapılandırma, tüm kullanıcıların Web sitesi veya uygulama içeriğine erişmesine izin veren varsayılan IIS yetkilendirme ayarlarını kaldırır ve ardından yalnızca ad-sorumlu kullanıcılara izin veren bir yetkilendirme kuralı yapılandırır.

**Örnek 11.12**

```
<konfigürasyon>
  <system.webServer>
    <güvenlik>
      <yetkilendirme>
        <remove users="*" roles="" verbs="" />
        <add accessType="Allow" users="" roles="Administrators" />
      </authorization>
    </güvenlik>
  </system.webServer>
</configuration
```

içeriğe erişmek için yönetici ayrıcalıkları.
IIS, gelen HTTP istekleri üzerinde sınırları zorlamak da dahil olmak üzere filtrelemeyi destekler. **Tablo 15,** ayarlanabilen IIS güvenlik parametrelerinin çoğunu göstermektedir.

**Tablo 15:** IIS Güvenlik Parametreleri

| Parametre | Fonksiyon |
|---|---|
| denyUrlSequences | Yasaklanmış URL kalıplarının bir listesi |
| fileExtensions | İzin verilen veya yasaklanan dosya uzantıları |
| hiddenSegments | Göz atılamayan URL'ler |
| requestLimits | URL, içerik, sorgu dizesi ve HTTP başlığı uzunluk sınırları |
| fiiller | İzin verilen veya yasaklanan fiiller |
| alwaysAllowedUrls | URL'lere her zaman izin verilir |
| alwaysAllowedQueryStrings | Sorgu dizelerine her zaman izin verilir |
| denyQueryStringSequences | Yasaklanmış sorgu dizeleri |
| filteringRules | Özel filtreleme kuralları |

Bu parametreler **<system.webServer><security><requestFiltering>** bölümünde yapılandırılır. **Örnek 11.13'teki örnek**:

- İki URL dizisine erişimi reddeder. İlk sekans dizin geçişini engeller ve ikinci sekans alternatif veri akışlarına erişimi engeller.

- Bir URL için maksimum uzunluğu 2KB ve bir sorgu dizesi için maksimum uzunluğu 1KB olarak ayarlar.

**Örnek 11.13**

```
<konfigürasyon>
  <system.webServer>
    <güvenlik>
      <requestFiltering>
        <denyUrlSequences>
          <add sequence=".." />
          <add sequence=":" />
        </denyUrlSequences>
        <fileExtensions allowUnlisted="false" />
        <requestLimits maxUrl="2048" maxQueryString="1024" />
        <verbs allowUnlisted="false" />
      </requestFiltering>
    </güvenlik>
  </system.webServer>
</configuration>
```

• Listelenmemiş dosya adı uzantılarına ve listelenmemiş HTTP fiillerine erişimi reddeder.
IIS, SSL'nin desteklenip desteklenmediğini, gerekli olup olmadığını, istemci kimlik doğrulamasının desteklenip desteklenmediğini ve şifre gücünü belirtmeye izin verir. <system.webServer><security><access> bölümünde yapılandırılır. Eski

**Örnek 11.14**

```
<location path="MySite">
  <system.webServer>
    <güvenlik>
      <access sslFlags="ssl">
    </güvenlik>
  </system.webServer>
</location>
```

**Şekil A5.13'**teki geniş alan MySite sitesine yapılan tüm bağlantılar için SSL'nin gerekli olduğunu belirtir.
IIS, kaynak IP adresleri veya DNS adları üzerinde kısıtlamalara izin verir. **Örnek 11.15**'te gösterildiği gibi **<system.webServer><se- curity><ipSecurity>** bölümünde yapılandırılır, burada örnek yapılandırma aşağıdakilere erişimi reddeder

**Örnek 11.15**

```
<location path="Varsayılan Web Sitesi">
  <system.webServer>
    <güvenlik>
      <ipSecurity>
        <add ipAddress="192.168.100.1" />
        <add ipAddress="169.254.0.0" subnetMask="255.255.0.0" />
      </ipSecurity>
    </güvenlik>
  </system.webServer>
</location>
```

IP adresi **192.168.100.1** ve tüm **169.254.0.0** ağına:
IIS güvenlik yapılandırması hakkında ayrıntılı bilgi IIS Güvenlik Yapılandırması bölümünde bulunabilir. Belirli güvenlik özelliği yapılandırma bilgileri Kimlik Doğrulama, Yetkilendirme, SSL, Kaynak IP, İstek Filtreleme ve Özel İstek Filtreleme bölümlerinde bulunabilir[**12].**

### 11.8 Programatik Yapılandırma: Microsoft IIS

Microsoft IIS güvenlik yapılandırması çeşitli dillerden programlı olarak da ayarlanabilir:

• appcmd.exe set config
• C#
• Visual Basic
• JavaScript

Örneğin, MySite adlı bir site için anonim kimlik doğrulamasını devre dışı bırakmak, ardından site için hem temel kimlik doğrulamasını hem de windows kimlik doğrulamasını etkinleştirmek (yukarıdaki bölümde yapılandırma yoluyla yapıldığı gibi)

**Örnek 11.16**

```
appcmd.exe set config "MySite" -section:system.webServer/security/authentication
    /anonymousAuthentication /enabled: "False" /commit:apphost
appcmd.exe set config "MySite" -section:system.webServer/security/authentication
   /basicAuthentication /enabled: "True" /commit:apphost
appcmd.exe set config "MySite" -section:system.webServer/security/authentication
  /windowsAuthentication /enabled: "True" /commit:apphost
```

şekil **örnek 11.16'**daki komutlar kullanılarak komut satırından gerçekleştirilir.

**Örnek 11.17**

```
using System;
using System.Text;
using Microsoft.Web.Administration;
internal static class Sample {

  private static void Main() {

    using(ServerManager serverManager = new ServerManager()) {
      Configuration config = serverManager.GetApplicationHostConfiguration();

      ConfigurationSection anonymousAuthenticationSection =
        config.GetSection("system.webServer/security/authentication
                  /anonymousAuthentication", "MySite");
      anonymousAuthenticationSection["enabled"] = false;

      ConfigurationSection basicAuthenticationSection =
        config.GetSection("system.webServer/security/authentication
                  /basicAuthentication", "MySite");
      basicAuthenticationSection["enabled"] = true;

      ConfigurationSection windowsAuthenticationSection =
        config.GetSection("system.webServer/security/authentication
                  /windowsAuthentication", "MySite");
      windowsAuthenticationSection["enabled"] = true;

      serverManager.CommitChanges();
    }
  }
}
```

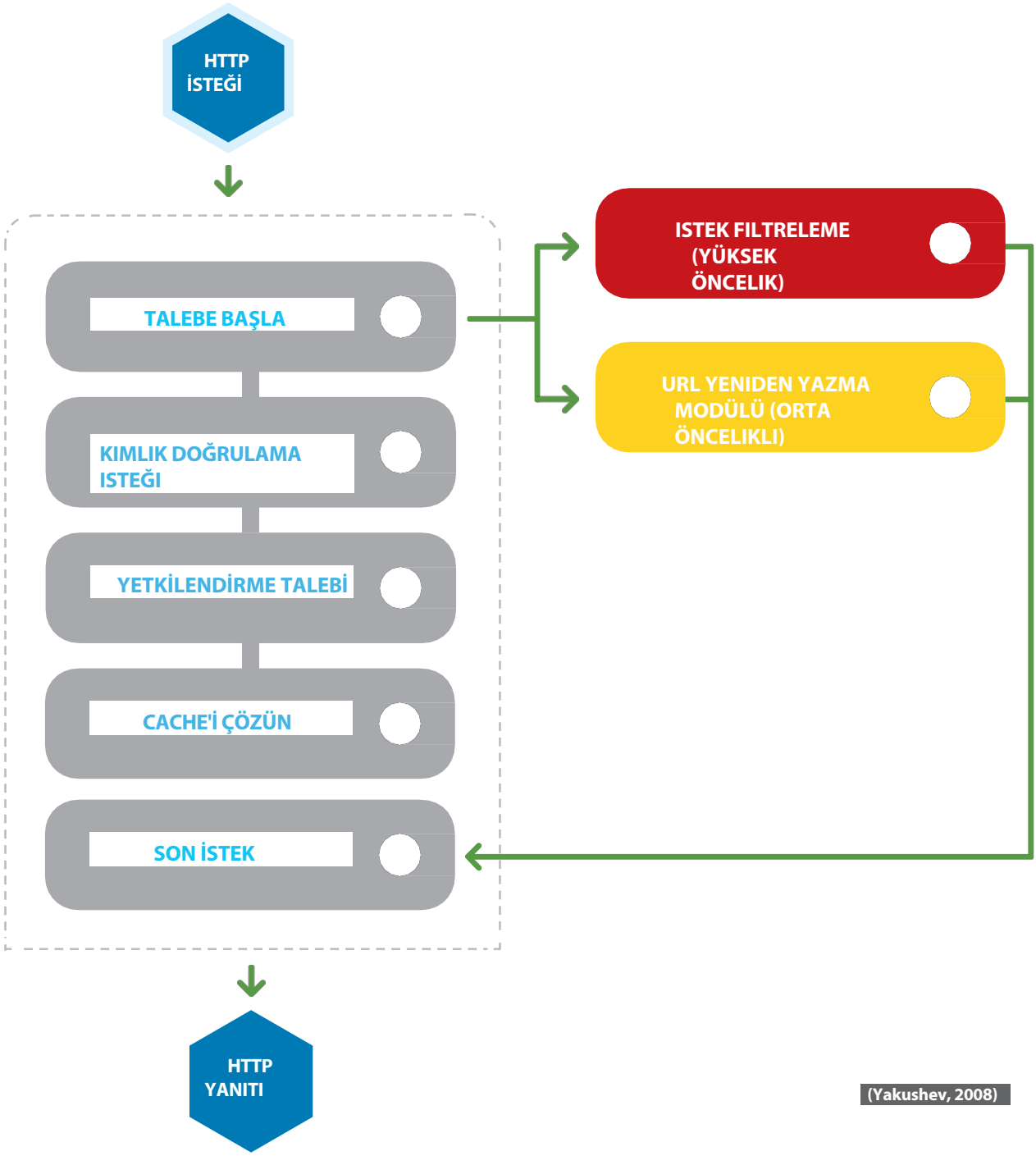
Alternatif olarak, aynı kimlik doğrulama kurulumu **örnek 11.17'**deki gibi programlı olarak kodlanabilir.
Kaynak kodu gözden geçirirken, güvenlik bölümlerindeki yapılandırma güncellemelerine özel dikkat gösterilmelidir.

## 11.9 Diğer IIS Yapılandırmaları

### İstekleri Filtreleme ve URL Yeniden Yazma

İstek Filtreleme IIS7'de tanıtıldı ve IIS 6.0 için UrlScan eklentisi işlevinin yerini aldı. Bu yerleşik güvenlik özelliği istenmeyen URL isteklerini filtrelemeye izin verir, ancak farklı filtreleme türlerini yapılandırmak da mümkündür. Başlangıç olarak, bir istek yapıldığında IIS işlem hattının nasıl çalıştığını anlamak önemlidir. Bu

**Şekil 8:** IIS İstek Filtreleme Dosyaları




(Yakushev, 2008)

aşağıdaki şema bu modüllerdeki sırayı göstermektedir

**Örnek 11.18**

```
<konfigürasyon>
  <system.webServer>
    <güvenlik>
      <requestFiltering>
        <denyUrlSequences>
          <add sequence=".." />
          <add sequence=":" />
        </denyUrlSequences>
        <fileExtensions allowUnlisted="false" />
        <requestLimits maxUrl="2048" maxQueryString="1024" />
        <verbs allowUnlisted="false" />
      </requestFiltering>
    </güvenlik>
  </system.webServer>
</configuration>
```

İstek filtreleme IIS arayüzü ya da web.config dosyası üzerinden ayarlanabilir. Örnek: (Yakushev, 2008)

**Örnek 11.19**

```
using System;
 using System.Text;
 using Microsoft.Web.Administration;
 internal static class Örnek
 {
   private static void Main()
   {
     using (ServerManager serverManager = new ServerManager())
     {
       Configuration config = serverManager.GetWebConfiguration("Default Web Site");
       ConfigurationSection requestFilteringSection = config.GetSection("system.webServer/security
       /requestFiltering");
      ConfigurationElementCollection denyUrlSequencesCollection =
      requestFilteringSection.GetCollection("denyUrlSequences");
      ConfigurationElement addElement = denyUrlSequencesCollection.CreateElement("add");
      addElement["sequence"] = @"...";
      denyUrlSequencesCollection.Add(addElement);
      ConfigurationElement addElement1 = denyUrlSequencesCollection.CreateElement("add");
      addElement1["sequence"] = @":";
      denyUrlSequencesCollection.Add(addElement1);
      ConfigurationElement addElement2 = denyUrlSequencesCollection.CreateElement("add");
      addElement2["sequence"] = @"\";
      denyUrlSequencesCollection.Add(addElement2);
```

```
        serverManager.CommitChanges();
      }
    }
  }
```

Bu, örneğin uygulama kodu aracılığıyla da yapılabilir:
(Yakushev, 2008)

**Çift Kodlu İstekleri Filtreleme**
Bu saldırı tekniği, güvenlik kontrollerini atlamak veya uygulamadan beklenmedik davranışlara neden olmak için kullanıcı istek parametrelerini onaltılık biçimde iki kez kodlamaktan oluşur. Bu mümkündür çünkü web sunucusu istemci isteklerini birçok kodlanmış biçimde kabul eder ve işler.

Çift kodlama kullanarak, kullanıcı girdisinin kodunu yalnızca bir kez çözen güvenlik filtrelerini atlamak mümkündür. İkinci kod çözme işlemi, kodlanmış verileri düzgün bir şekilde işleyen ancak ilgili güvenlik kontrollerine sahip olmayan arka uç platformu veya modülleri tarafından yürütülür.

Saldırganlar, web uygulaması tarafından kullanılan kimlik doğrulama şemasını ve güvenlik filtrelerini atlamak için yol adlarına veya sorgu dizelerine çift kodlama ekleyebilir.

Web uygulamaları saldırılarında kullanılan bazı yaygın karakterler vardır. Örneğin, Path Traversal saldırılarında **"../" (dot-dot-slash) karakterini** kullanırken, XSS saldırıları "<" ve ">" karakterlerini kullanır. Bu karakterler normal verilerden farklı bir onaltılık gösterim verir.

Örneğin, **"../" (nokta-nokta-eğik çizgi)** karakterleri onaltılık gösterimde **%2E%2E%2f'yi** temsil eder. Ne zaman sembolü tekrar kodlanır, onaltılık koddaki gösterimi **%25**'tir. Çift kodlama işleminden elde edilen sonuç **"../" (nokta-nokta-eğik çizgi) %252E%252E%252F** olacaktır:

- "../" onaltılık kodlaması "%2E%2E%2f "yi temsil eder
- Ardından "%" kodlaması "%25 "i temsil eder
- "../" çift kodlaması "%252E%252E%252F "yi temsil eder

IIS'nin iki kat kodlanmış isteklerin sunulmasına izin vermesini istemiyorsanız, aşağıdakileri kullanın (IIS Ekibi,2007):

**Örnek 11.20**

```
<konfigürasyon>
  <system.webServer>
   <güvenlik>
    <requestFiltering
     allowDoubleEscaping="false">
    </requestFiltering>
   </güvenlik>
  </system.webServer>
 </configuration>
```

**Yüksek Bit Karakterleri Filtreleme**
Bu, IIS'ye yapılan ve ASCII olmayan karakterler içeren tüm isteklere izin verir veya bunları reddeder. Bu gerçekleştiğinde kullanıcıya 404.12. hata kodu görüntülenir. **UrlScan (IIS6 eklentisi)** eşdeğeri **AllowHighBitCharacters'dır**.

**Örnek 11.21**

```
<konfigürasyon>
 <system.webServer>
  <güvenlik>
   <requestFiltering
   allowHighBitCharacters="true">
   </requestFiltering>
  </güvenlik>
 </system.webServer>
</configuration>
```

**Dosya Uzantılarına Göre Filtreleme**
Bu filtreyi kullanarak IIS'nin dosya uzantılarına dayalı bir isteğe izin vermesini sağlayabilirsiniz, kaydedilen hata kodu 404.7'dir. **Al- lowExtensions** ve **DenyExtensions** seçenekleri **UrlScan** eşdeğerleridir.

**Örnek 11.22**

```
<konfigürasyon>
 <system.webServer>
  <güvenlik>
   <requestFiltering>
    <fileExtensions allowUnlisted="true">
    <add fileExtension=".asp" allowed="false"/>
    </fileExtensions>
   </requestFiltering>
  </güvenlik>
 </system.webServer>
</configuration>
```

**İstek Sınırlarına Göre Filtreleme**
IIS bir isteği istek sınırlarına göre reddettiğinde, günlüğe kaydedilen hata kodu şöyledir:
- 404.13 içerik çok uzunsa.
- 404.14 URL çok büyükse.
- 404.15 sorgu dizesi çok uzunsa.

Bu, sorunu çözmek için kaynak kodunu değiştiremeyeceğiniz bir uygulamaya gönderilen uzun bir sorgu dizesini veya çok fazla içeriği sınırlamak için kullanılabilir.

**Örnek 11.23**

```
<konfigürasyon>
<system.webServer>
 <güvenlik>
  <requestFiltering>
   <requestLimits
    maxAllowedContentLength="30000000"
    maxUrl="260"
    maxQueryString="25"
    />
  </requestFiltering>
 </güvenlik>
</system.webServer>
</configuration>
```

**Fiillere Göre Filtrele**

IIS bu özelliğe dayanarak bir isteği reddettiğinde, günlüğe kaydedilen hata kodu 404.6'dır. Bu, UrlScan'deki UseAl- lowVerbs, AllowVerbs ve DenyVerbs seçeneklerine karşılık gelir.
Uygulamanın yalnızca belirli fiil türlerini kullanmasını istiyorsanız, önce allowUnlisted'i 'false' olarak ayarlamanız ve ardından izin vermek istediğiniz fiilleri ayarlamanız gerekir (örneğe bakın)

**Örnek 11.24**

```
<konfigürasyon>
  <system.webServer>
   <güvenlik>
    <requestFiltering>
     <verbs allowUnlisted="false">
      <add verb="GET" allowed="true" />
     </verbs>
    </requestFiltering>
   </güvenlik>
  </system.webServer>
 </configuration>
```

**URL Sıralarına Göre Filtreleme**

Bu özellik, bir isteğin parçası olduğunda IIS'nin reddedebileceği dizilerin bir listesini tanımlar. IIS bu özellik için bir isteği reddettiğinde, günlüğe kaydedilen hata kodu 404.5'tir. Bu, UrlScan'deki DenyUrlSequences özelliğine karşılık gelir. Bu çok güçlü bir özelliktir. Bu, belirli bir karakter dizisinin IIS tarafından hiç katılmasını önler:

**Örnek 11.25**

```
<konfigürasyon>
 <system.webServer>
```

```
    <güvenlik>
     <requestFiltering>
      <denyUrlSequences>
       <add sequence=".."/>
      </denyUrlSequences>
     </requestFiltering>
    </güvenlik>
   </system.webServer>
  </configuration>
```

**Gizli Segmentleri Filtreleme**
IIS'in ikili dizindeki içeriği sunmasını ancak ikiliyi sunmamasını istiyorsanız, bu yapılandırmayı uygulayabilirsiniz.

**Örnek 11.26**

```
<konfigürasyon>
  <system.webServer>
    <güvenlik>
     <requestFiltering>
      <hiddenSegments>
       <add segment="BIN"/>
      </hiddenSegments>
     </requestFiltering>
    </güvenlik>
   </system.webServer>
  </configuration>
```

**Parola koruması ve hassas bilgiler**
Web.config dosyaları, bağlantı dizelerinde veritabanı parolaları, posta sunucusu kullanıcı adları gibi hassas bilgiler içerebilir.

Şifrelenmesi gereken bölümler şunlardır:
- <appSettings>. Bu bölüm özel uygulama ayarlarını içerir.
- <connectionStrings>. Bu bölüm bağlantı dizelerini içerir.
- <kimlik>. Bu bölüm kimliğe bürünme kimlik bilgilerini içerebilir.
- <sessionState>. Bu bölüm, işlem dışı oturum durumu sağlayıcısı için bağlantı dizesini içerir.

Bir <connectionstring> bölümünde bulunan parolalar ve kullanıcı adları şifrelenmelidir. ASP.NET, aspnet_regiis işlevini kullanarak bu bilgileri şifrelemenize olanak tanır.
NET framework klasörü altında
**windows%\Microsoft.NET\Framework\v2.0.50727**

Komutunu kullanarak şifrelemeniz gereken bölümü belirtebilirsiniz:
**aspnet_regiis -pef sectiontobeencryoted**

**Web.Config dosyasındaki bölümleri şifreleme**
Bölümlerin şifrelenmesi mümkün olsa da, tüm bölümler şifrelenemez, özellikle de

kullanıcı kodu çalıştırılmadan önce okunur. Aşağıdaki bölümler şifrelenemez:

- <processModel>
- <çalışma zamanı>
- <mscorlib>
- <başlangıç>
- <system.runtime.remoting>
- <configProtectedData>
- <uydu montajları>
- <cryptographySettings>
- <cryptoNameMapping>
- <cryptoClasses>

**Makine Düzeyinde RSA anahtar konteyneri veya Kullanıcı Düzeyinde Anahtar Konteynerleri**
Makine düzeyinde RSA anahtarı kullanarak tek bir dosyayı şifrelemenin, bu dosya başka sunuculara taşındığında dezavantajları vardır. Bu durumda, kullanıcı düzeyinde RSA anahtar konteyneri şiddetle tavsiye edilir. RSAProtectedConfigurationProvider, anahtar depolama için makine düzeyinde ve kullanıcı düzeyinde anahtar kaplarını destekler.

RSA makine anahtarı konteynerleri aşağıdaki klasörde saklanır:
**\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA\MachineKeys**

**Kullanıcı Anahtarı Konteyneri**
Korunması gereken uygulama paylaşılan bir barındırma ortamındaysa ve hassas verilerin korunmasına diğer uygulamalar erişemiyorsa, kullanıcı anahtarı konteyneri şiddetle tavsiye edilir.
Bu durumda her uygulama ayrı bir kimliğe sahip olmalıdır. RSA
kullanıcı seviyesi anahtar konteynerleri aşağıdaki klasörde
saklanır:

**\Documents and Settings\{UserName}\Application Data\Microsoft\Crypto\RSA**

**IIS yapılandırmaları**
Yapılandırılması gereken IIS sürümüne bağlı olarak, sunucuda güvenliği oluşturabilecek bazı ayarların gözden geçirilmesi önemlidir.

**Güven seviyesi**
Güven düzeyi, bir barındırma ortamında bir uygulamaya verilen Kod Erişim Güvenliği izinleri kümesidir. Bunlar ilke dosyaları kullanılarak tanımlanır. Yapılandırılması gereken güven düzeyine bağlı olarak, FULL, HIGH, MEDIUM, LOW veya MINIMAL düzeylerinde izin vermek mümkündür. ASP.NET ana bilgisayarı, tam güven düzeyinde çalışan uygulamalara herhangi bir ek ilke uygulamaz.

Örnek:

**Örnek 11.27**

```
<sistem.web>
  <securityPolicy>
    <trustLevel name="Full" policyFile="internal"/>
  </securityPolicy>
 </system.web>
```

**Güven Seviyelerini Kilitle**
NET framework web.config dosyasında, uygulamaların güven düzeylerini değiştirmelerini kilitlemek mümkündür Bu dosya şu adreste bulunur:

**C:\Windows\Microsoft.NET\Framework\{version}\CONFIG**

Aşağıdaki örnekte 2 farklı uygulama yapılandırması güven düzeyinin nasıl kilitleneceği gösterilmektedir (MSDN, 2013)

**Örnek 11.28**

```
<konfigürasyon>
  <location path="application1" allowOverride="false">
    <sistem.web>
      <güven seviyesi="Yüksek" />
    </system.web>
  </location>
  <location path="application2" allowOverride="false">
    <sistem.web>
      <güven seviyesi="Orta" />
    </system.web>
  </location>
 </configuration>
```

**Referanslar**

• Yakushev Ruslan , 2008 "IIS 7.0 Request Filtering and URL Rewriting " **http://www.iis.net/learn/extensions/url-rewrite-module/iis-request-filtering-and-url-rewriting** adresinden erişilebilir (Son erişim tarihi 14 Temmuz, 2013)

• OWASP, 2009 "Double Encoding" https://www.owasp.org/index.php/Double_Encoding adresinde mevcuttur (Son erişim tarihi 14 Temmuz 2013)

• IIS Ekibi, 2007 "İstek Filtrelemeyi Kullanın " **http://www.iis.net/learn/manage/configuring-security/ use-request-filtering** adresinden erişilebilir (Son erişim tarihi 14 Temmuz 2013)

• Aguilar Carlos ,2006 "The new Configuration System in IIS 7" **http://blogs.msdn.com/b/carlosag/archive/2006/04/25/iis7configurationsystem.aspx** adresinden erişilebilir (Son erişim tarihi 14 Temmuz 2013)

• MSDN, 2013 . Nasıl Yapılır? ASP.NET Yapılandırma Ayarlarını Kilitleme **http://msdn.microsoft.com/en-us/library/ms178693.aspx** adresinde bulunabilir (Son erişim tarihi 14 Temmuz 2013)

## 11.10 Güçlü Adlandırılmış Meclisler

Derleme işlemi sırasında QA ya da Geliştiriciler kodu çalıştırılabilir formatlarda yayınlayacaktır. Genellikle bu bir exe veya bir ya da birkaç DLL'den oluşur. Derleme/yayınlama işlemi sırasında kodun imzalanması ya da imzalanmaması yönünde bir karar verilmesi gerekir.

Kodunuzu imzalamak Microsoft tarafından "güçlü isimler" oluşturmak olarak adlandırılır. Visual Studio kullanarak bir proje oluşturur ve Microsoft'un "Kod analizini çalıştır" özelliğini kullanırsanız, kod güçlü bir şekilde adlandırılmamışsa büyük olasılıkla bir Microsoft tasarım hatasıyla karşılaşırsınız; "Uyarı 1 CA2210 : Microsoft.Design : 'xxx.exe' dosyasını güçlü bir ad anahtarıyla imzala."

Kod incelemesi, güçlü adlandırma kullanılıp kullanılmadığının, faydalarının ve güçlü adlandırmanın hangi tehdit vektörlerini önlemeye yardımcı olduğunun farkında olmalı veya güçlü adlandırma kullanmamanın nedenlerini anlamalıdır.

Güçlü ad, bir derlemin kimliğini metin adını, sürüm numarasını, kültür bilgisini, açık anahtarı ve dijital imzayı kullanarak imzalama yöntemidir. (Solis, 2012)

• Güçlü adlandırma, bu montaj için benzersiz bir ad garanti eder.

• Güçlü adlar bir derlemenin sürüm soyağacını korur. Güçlü bir ad, hiç kimsenin derlemenizin sonraki bir sürümünü oluşturamamasını sağlayabilir. Kullanıcılar, yükledikleri derlemenin bir sürümünün, uygulamanın oluşturulduğu sürümü oluşturan aynı yayıncıdan geldiğinden emin olabilirler.

Global Assembly Cache (GAC) kullanacaksanız yukarıdaki iki nokta çok önemlidir.

• Güçlü adlar güçlü bir bütünlük denetimi sağlar ve sahteciliği önler. .NET Framework güvenlik denetimlerini geçmek, derlemenin içeriğinin oluşturulmasından bu yana değiştirilmediğini garanti eder.

Ancak, güçlü adların kendi başlarına, örneğin dijital imza ve destekleyici sertifika tarafından sağlanan gibi bir güven düzeyi anlamına gelmediğini unutmayın. GAC derlemelerini kullanıyorsanız, tasarım gereği GAC kilitli, yalnızca yöneticilere özel bir depo olduğundan, derlemelerin her yüklendiklerinde doğrulanmadığını unutmayın.

Güçlü adların engelleyemediği şey, kötü niyetli bir kullanıcının güçlü ad imzasını tamamen silmesi, derlemeyi değiştirmesi veya kötü niyetli kullanıcının anahtarıyla yeniden imzalamasıdır.

Kodu gözden geçiren kişinin güçlü ad özel anahtarının nasıl güvende tutulacağını ve yönetileceğini anlaması gerekir. Güçlü ad imzalarının kuruluşunuz için uygun olduğuna karar verirseniz bu çok önemlidir.

En az ayrıcalık ilkesi kullanılırsa, kod bilgisayar korsanlarının erişimine açık olmaz veya daha az açık olur ve GAC kullanılmazsa güçlü isimler daha az fayda sağlar veya hiç fayda sağlamaz.

**Güçlü Adlandırma Nasıl Kullanılır**

İmzalama araçları

Güçlü bir isim grubu oluşturmak için izlemeniz gereken bir dizi araç ve adım vardır

Visual Studio Kullanımı

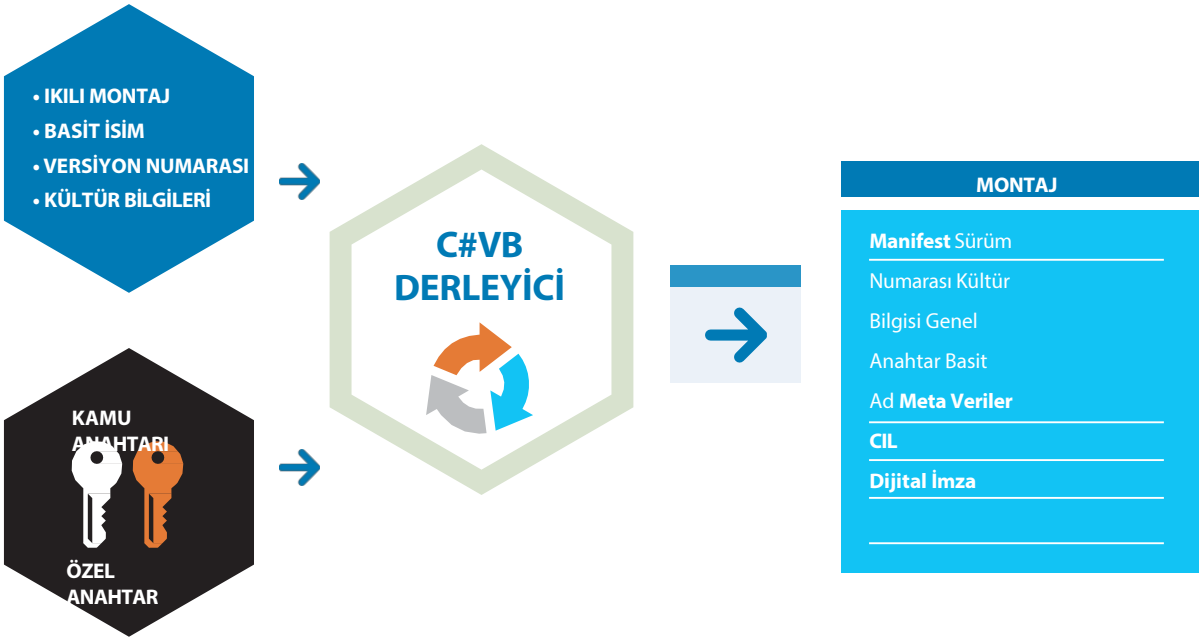
Visual Studio'yu kullanarak bir Güçlü Adlandırılmış Derleme oluşturmak için genel/özel anahtar çifti dosyasının bir kopyasına sahip olmak gerekir. Bu anahtar çiftini Visual Studio'da oluşturmak da mümkündür

Visual Studio 2005'te, C#, Visual Basic ve Visual J# tümleşik geliştirme ortamları (IDE'ler), Sn.exe (Güçlü Ad Aracı) kullanarak anahtar çifti oluşturmanıza gerek kalmadan anahtar çiftleri oluşturmanıza ve derlemeleri imzalamanıza olanak tanır.

Bu IDE'lerin Proje Tasarımcısında bir İmzalama sekmesi vardır. . Anahtarları tanımlamak için AssemblyKeyFileAttribute kullanımı
dosya çiftleri Visual Studio 2005'te kullanılmaz hale getirilmiştir.

Aşağıdaki şekilde derleyici tarafından yapılan işlem gösterilmektedir

**Şekil 9:** C# Güçlü Adlandırma




**Güçlü Ad aracını kullanma**

İmzalama Aracı, dosyaları dijital olarak imzalayan, dosyalardaki imzaları doğrulayan veya dosyalara zaman damgası vuran bir komut satırı aracıdır. İmzalama Aracı Microsoft Windows NT, Windows Me, Windows 98 veya Windows 95 üzerinde desteklenmez.

"Visual Studio Komut İstemi" (Başlat >> Microsoft Visual Studio 2010 >> Visual Studio Araçları >> Visual Studio Komut İstemi (2010)) kullanmıyorsanız **sn.exe** dosyasını **%ProgramFiles%\Microsoft SDKs\Windows\v7.0A\bin\sn.exe** adresinde bulabilirsiniz

Aşağıdaki komut yeni, rastgele bir anahtar çifti oluşturur ve bunu keyPair.snk dosyasında saklar.
**sn -k keyPair.snk**

Aşağıdaki komut keyPair.snk dosyasındaki anahtarı MyContainer konteynerinde CSP güçlü adıyla saklar.
**sn -i keyPair.snk MyContainer**

Aşağıdaki komut keyPair.snk dosyasından açık anahtarı çıkarır ve publicKey.snk dosyasında saklar.
**sn -p keyPair.snk publicKey.snk**

Aşağıdaki komut publicKey.snk dosyasında bulunan açık anahtarı ve açık anahtarın belirtecini görüntüler.
**sn -tp publicKey.snk**

Aşağıdaki komut MyAsm.dll derlemesini doğrular.
**sn -v MyAsm.dll**

Aşağıdaki komut MyContainer'ı varsayılan CSP'den siler.
**sn -d MyContainer**

**Montaj Bağlayıcısını (Al.exe) Kullanma**

Bu araç Visual Studio ve Windows SDK ile birlikte otomatik olarak yüklenir. Aracı çalıştırmak için Visual Studio Komut İstemi veya Windows SDK Komut İstemi'ni (CMD Shell) kullanmanızı öneririz. Bu yardımcı programlar, kurulum klasörüne gitmeden aracı kolayca çalıştırmanızı sağlar. Daha fazla bilgi için Visual Studio ve Windows SDK Komut İstemleri bölümüne bakın.

**Bilgisayarınızda Visual Studio yüklüyse:**
Görev çubuğunda Başlat'ı tıklatın, Tüm Programlar'ı tıklatın, Visual Studio'yu tıklatın, Visual Studio Araçları'nı tıklatın ve ardından Visual Studio Komut İstemi'ni tıklatın.
-ya da-
**Bilgisayarınızda Windows SDK yüklüyse:**
Görev çubuğunda Başlat'ı, Tüm Programlar'ı, Windows SDK klasörünü ve ardından Komut İstemi'ni (veya CMD Kabuğu) tıklatın.

Komut istemine aşağıdakileri yazın:
**al kaynak seçenekleri**

**Açıklamalar**
Tüm Visual Studio derleyicileri derlemeler üretir. Ancak bir veya daha fazla modülünüz (manifestosu olmayan meta veriler) varsa, manifestosu ayrı bir dosyada olan bir montaj oluşturmak için Al.exe'yi kullanabilirsiniz. Önbelleğe montaj yüklemek, önbellekten montaj kaldırmak veya önbellek içeriğini listelemek için Global Assembly Cache Tool'u (**Gacutil.exe**) kullanın.

Aşağıdaki komut, t2.netmodule modülünden bir montaj ile t2a.exe yürütülebilir dosyasını oluşturur. Giriş noktası **MyClass** içindeki Main yöntemidir.
**al t2.netmodule /target:exe /out:t2a.exe /main:MyClass.Main**

**Montaj özniteliklerini kullanma**
Güçlü ad bilgisini doğrudan koda ekleyebilirsiniz. Bunun için, anahtar dosyasının bulunduğu yere bağlı olarak AssemblyKeyFileAttribute veya AssemblyKeyNameAttribute kullanabilirsiniz

**Derleyici seçeneklerini kullanın :use /keyfile veya /delaysign**
Anahtar çiftinin geliştiricilerden korunması, bir araya getirmelerin bütünlüğünü korumak ve garanti altına almak için gereklidir. Genel anahtar erişilebilir olmalıdır, ancak özel anahtara erişim yalnızca birkaç kişiyle sınırlıdır. Güçlü isimlere sahip montajlar geliştirirken, güçlü isimli hedefe referans veren her montaj, hedef montaja güçlü bir isim vermek için kullanılan ortak anahtarın token'ını içerir. Bu, geliştirme süreci boyunca ortak anahtarın kullanılabilir olmasını gerektirir.

Güçlü ad imzası için taşınabilir yürütülebilir (PE) dosyasında yer ayırmak, ancak asıl imzalamayı daha sonraki bir aşamaya (genellikle montajı göndermeden hemen önce) ertelemek için derleme zamanında gecikmeli veya kısmi imzalama kullanabilirsiniz.
**C# ve VB.NET'te /keyfile veya /delaysign kullanabilirsiniz (MSDN)**

**Referanslar**
- http://msdn.microsoft.com/en-us/library/wd40t7ad(v=vs.80).aspx
- http://msdn.microsoft.com/en-us/library/c405shex(v=vs.110).aspx
- http://msdn.microsoft.com/en-us/library/k5b5tt23(v=vs.80).aspx

• http://msdn.microsoft.com/en-us/library/t07a3dye(v=vs.80).aspx

• http://msdn.microsoft.com/en-us/library/t07a3dye(v=vs.110).aspx

## 11.10 Yuvarlak Açma

Round Tripping, saldırganın belirli bir uygulamadan bir montajı derlemesini sağlayan bir tersine mühendislik tekniğidir. Ildasm.exe bu amaçla kullanılabilir ve ILAsm derlemeyi yeniden derlemek için kullanılır.

MSIL Disassembler (Ildasm.exe), MSIL Assembler'a (Ilasm.exe) eşlik eden bir araçtır. Ildasm.exe, Microsoft ara dil (MSIL) kodu içeren taşınabilir bir yürütülebilir (PE) dosya alır ve Ilasm.exe'ye girdi olarak uygun bir metin dosyası oluşturur. Bu araç Visual Studio ve Windows SDK ile birlikte otomatik olarak yüklenir.

**Gizlemenin Önemi**

Daha önce de belirtildiği gibi, Round Tripping, derlemelere tersine mühendislik uygulamak için kullanılan bir tekniktir. Bu nedenle, derlemelerinizin tersine mühendisliğe tabi tutulmasını veya daha da kötüsü, kodun Ildasm ve Ilasm araçları kullanılarak kötü niyetli manipülasyonun kurbanı olmasını önlemek istiyorsanız, bunu uygulamanız önerilir. Bu amaçla kullanılabilecek DeepSea, Crypto veya Dotfuscator gibi farklı türde ürünler vardır.

**Karartma Kullanma**

Tersine mühendislik ve derlemelerin kurcalanmasını önlemek için kullanılan en etkili teknik, Karartma kullanımıdır. Visual Studio, Dotfuscator'ın bir sürümünü içerir. Bu programa VS menüsünden Araçlar Dotfuscator (Community Edition menü komutu) seçilerek erişilebilir. Not: Bu araçlar Express sürümlerinde mevcut değildir

Montajlarınızı gizlemek için:
• Projeyi VS Studio'da oluşturun
• Araçlar--> Dotfuscator Topluluk Sürümü
• Bir ekran hangi proje türünü sorar, 'Yeni Proje Oluştur'u seçin ve Tamam'a tıklayın
• Dotfuscator arayüzünün Giriş sekmesinde, 'Gözat ve Listeye montaj ekle'

seçeneğine tıklayın Derlenmiş uygulamaya gözatın

**ASPNetConfigs**

Giriş

ASP.NET uygulamalarında kaynakların güvenliğini sağlamak Web.config dosyasındaki yapılandırma ayarlarının bir kombinasyonudur, ancak IIS yapılandırmalarının da bunda büyük bir rol oynadığını unutmamak önemlidir. Bu, toplam bir güvenlik çerçevesi sağlayan entegre bir yaklaşımdır.
Aşağıda, web.config dosyası içindeki ASP.NET yapılandırma ayarlarının en önemli yönleri vurgulanmaktadır. Tam bir genel bakış için ASP.NET güvenliği bölümüne bakın (https://www.owasp.org/index.php/CRV2_Framework- SpecIssuesASPNet)

**Güvenli Yapılandırma Değerleri**

Yapılandırma dosyalarına kaydedilen Hassas Bilgiler şifrelenmelidir. Örneğin machineKey öğesinde saklanan şifreleme anahtarları veya veritabanında oturum açmak için kullanıcı adı ve parolaları içeren bağlantı dizeleri.

**ASP.NET Yapılandırma ayarlarını kilitleme**

ASP.NET yapılandırma dosyalarındaki (Web.config dosyaları) yapılandırma ayarlarını bir konum öğesine allowOver- ride özniteliği ekleyerek kilitleyebilirsiniz

**Konum Ayarlarını kullanarak dizinleri yapılandırma**
<location> öğesi aracılığıyla belirli klasörler ve dosyalar için ayarlar oluşturabilirsiniz. Path niteliği, dosyayı veya alt dizini belirtmek için kullanılır. Bu, Web.config dosyası örneğinde yapılmıştır:

**Örnek 11.29**

```
<location path="."> 
   <bölüm1 .../> 
   <bölüm2 ... /> 
 </location> 
 <location path="Varsayılan Web Sitesi"> 
  <bölüm1 ... /> 
  <bölüm2 ... /> 
 </location 
 <location path="Varsayılan Web Sitesi/MyApplication/Admin/xyz.html"> 
  <bölüm1 ... /> 
  <bölüm2 ... /> 
 </location>
```

**Hata Kodu işleme için istisnaları yapılandırma**
Bir kullanıcı hatalı bir istek veya geçersiz parametreler gönderdiğinde doğru hata kodunun gösterilmesi ve işlenmesi önemli bir yapılandırma konusudur. Bu hataların günlüğe kaydedilmesi, uygulamaya yönelik olası saldırıları analiz ederken de mükemmel bir yardımcıdır.

Bu hataları kodda veya Web.Config dosyasında yapılandırmak mümkündür

HttpException yöntemi, HTTP isteklerinin işlenmesi sırasında meydana gelen bir istisnayı açıklar:

**Örnek 11.30**

```
if (string.IsNullOrEmpty(Request["id"])) 
    throw new HttpException(400, "Bad request");
```

veya Web.config dosyasında:

**Örnek 11.31**

```
<konfigürasyon> 
 <sistem.web> 
  <customErrors mode="On" defaultRedirect="ErrorPage.html" 
          redirectMode="ResponseRewrite">
```

```
    <error statusCode="400" redirect="BadRequest.html" />
    <error statusCode="404" redirect="FileNotFound.html" />
   </customErrors>
  </system.web>
 </configuration>
```

**Girdi doğrulama**

Harici kaynaklardan gelen her şey bir web uygulamasında girdi olarak değerlendirilebilir. Yalnızca kullanıcının bir web formu aracılığıyla veri girmesi değil, aynı zamanda bir web hizmetinden veya veritabanından alınan veriler, tarayıcılardan gönderilen başlıklar da bu kavramın kapsamına girer. Girdinin ne zaman güvenli olduğunu tanımlamanın bir yolu, bir güven sınırı belirlemektir.

Güven sınırı olarak bilinen şeyi tanımlamak, tüm olası güvenilmeyen girdileri görselleştirmemize yardımcı olabilir. Bunlardan biri de kullanıcı girdisidir.ASP.NET, uygulanacak kontrol seviyesine bağlı olarak farklı doğrulama türlerine sahiptir. Varsayılan olarak, web sayfalarının kodu kötü niyetli kullanıcılara karşı doğrulanır. Aşağıda kullanılan doğrulama türlerinin bir listesi bulunmaktadır (MSDN, 2013):

**Şekil 10:** IIS Giriş Doğrulaması

| Doğrulama türü | Kullanılacak kontrol | Açıklama |
|---|---|---|
| Gerekli giriş | RequiredFieldValidator | Kullanıcının bir girişi atlamamasını sağlar. |
| Bir değerle karşılaştırma | CompareValidator | Kullanıcının girdisini sabit bir değerle, başka bir kontrolün değeriyle (küçük, eşit veya büyük gibi bir karşılaştırma operatörü kullanarak) veya belirli bir veri türüyle karşılaştırır. |
| Menzil kontrolü | RangeValidator | Bir kullanıcının girişinin belirtilen alt ve üst sınırlar arasında olup olmadığını kontrol eder. Sayı, alfabetik karakter ve tarih çiftleri içindeki aralıkları kontrol edebilirsiniz. |
| Desen eşleştirme | RegularExpressionValidator | Girdinin düzenli ifade tarafından tanımlanan bir desenle eşleşip eşleşmediğini denetler. Bu doğrulama türü, e-posta adresleri, telefon numaraları, posta kodları vb. gibi öngörülebilir karakter dizilerini denetlemenizi sağlar. |
| Kullanıcı tanımlı | CustomValidator | Kendi yazdığınız doğrulama mantığını kullanarak kullanıcının girişini kontrol eder. Bu doğrulama türü, çalışma zamanında türetilen değerleri kontrol etmenizi sağlar. |

## Referanslar

MSDN, 2013 "ASP.NET Yapılandırmalarının Güvenliğini Sağlama" şu adresten erişilebilir
http://msdn.microsoft.com/en-us/library/ms178699%28v=vs.100%29.aspx (Son Görüntüleme, 25 Temmuz 2013)

## 11.11 .NET Kimlik Doğrulama Kontrolleri

NET'te yapılandırma dosyasında Authentication etiketleri vardır. <authentication> öğesi, uygulamalarınızın kullandığı kimlik doğrulama modunu yapılandırır. Uygun kimlik doğrulama modu, uygulamanızın veya Web hizmetinizin nasıl tasarlandığına bağlıdır. Varsayılan Machine.config ayarı, aşağıda gösterildiği gibi güvenli bir Win-dows kimlik doğrulama varsayılanı uygular

**authentication Attributes:mode="[Windows|Forms|Passport|None]"**

**<kimlik doğrulama modu="Windows" />**

**Formlar Kimlik Doğrulama Yönergeleri**
Forms kimlik doğrulamasını kullanmak için <authentication> öğesinde mode="Forms" olarak ayarlayın. Sonraki, alt <forms> öğesini kullanarak Forms kimlik doğrulamasını yapılandırın. Aşağıdaki parça güvenli bir <forms> kimlik doğrulama öğesi yapılandırması:

**Örnek 11.32**

```
<Kimlik doğrulama modu="Formlar">
 <forms loginUrl="Restricted\login.aspx" SSL korumalı bir klasörde oturum açma sayfası
  protection="All" Gizlilik ve bütünlük
  requireSSL="true" Çerezin http üzerinden gönderilmesini
  önler timeout="10" Sınırlı oturum ömrü
  name="AppNameCookie" Uygulama başına benzersiz ad
  path="/FormsAuth" ve path
  slidingExpiration="true" > Oturum ömrünü kaydırma
 </forms>
</authentication>
```

Forms kimlik doğrulama güvenliğini artırmak için aşağıdaki önerileri kullanın:
- Web sitenizi bölümlere ayırın.
- Koruma="Tümü" olarak ayarlayın.
- Küçük çerez zaman aşımı değerleri kullanın.
- Sabit bir sona erme süresi kullanmayı düşünün.
- Forms kimlik doğrulaması ile SSL kullanın.
- SSL kullanmıyorsanız slidingExpiration = "false" olarak ayarlayın.
- <credentials> öğesini üretim sunucularında kullanmayın.
- <machineKey> öğesini yapılandırın.
- Benzersiz çerez adları ve yolları kullanın.

**Klasik ASP**
Klasik ASP sayfaları için kimlik doğrulama genellikle kullanıcı bilgilerinin bir DB'ye karşı doğrulamadan sonra ses değişkenlerine dahil edilmesiyle manuel olarak gerçekleştirilir, bu nedenle aşağıdaki gibi bir şey arayabilirsiniz:
Session ("UserId") = UserName
Session ("Roles") = UserRoles

**Kod İnceleme .Net Kod Yönetme**
NET Yönetilen kodu, yönetilmeyen kodda bulunan Tampon Taşmaları ve bellek bozulması gibi yaygın güvenlik açıklarına karşı daha az savunmasızdır, ancak kodda performansı ve güvenliği etkileyebilecek sorunlar olabilir. Aşağıda, kod incelemesi sırasında bakılması önerilen uygulamaların bir özeti yer almaktadır. Ayrıca, bu bölümde çalışmayı kolaylaştırabilecek ve kodunuzdaki kusurları anlamanıza ve tespit etmenize yardımcı olabilecek bazı araçlardan da bahsetmek gerekir

**Kod Erişim Güvenliği**
Bu, yarı güvenilir kodun yürütülmesini destekleyerek çeşitli güvenlik tehditlerini önler. Aşağıda, Kod Erişimi güvenliğinin yanlış kullanımından kaynaklanan olası güvenlik açıklarının bir özeti yer almaktadır:

**Tablo 16:** Kod Erişimi Güvenlik Açıkları

| Güvenlik Açığı | Çıkarımlar |
|---|---|
| Bağlantı taleplerinin veya iddialarının uygunsuz kullanımı | Kod, tuzak saldırılarına karşı hassastır |
| Kod, güvenilmeyen arayanlara izin verir | Kötü amaçlı kod, hassas işlemler gerçekleştirmek ve kaynaklara erişmek için kodu kullanabilir |

**Bildirimsel güvenlik**
Mümkün olduğunca zorunluluk yerine bildirimsel güvenlik kullanın.
Bildirimsel sözdizimi örneği(MSDN[2], 2013):

**Örnek 11.33**

```
[MyPermission(SecurityAction.Demand, Unrestricted = true)]
  public class MyClass
  {
   public Benim Sınıfım()
   {
     //Yapıcı güvenlik çağrısı tarafından korunur.
   }
   public void MyMethod()
   {
     //Bu yöntem güvenlik çağrısı tarafından korunmaktadır.
   }
   public void YourMethod()
   {
     //Bu yöntem güvenlik çağrısı tarafından korunmaktadır.
   }
}
```

**Yönetilmeyen kod**
C# güçlü tipli bir dil olmasına rağmen, 'unsafe' kodunu kullanarak yönetilmeyen kod çağrılarını kullanmak mümkündür. "Yöntem çağrıları arasında veritabanı bağlantısı gibi yönetilmeyen bir kaynak kullanan herhangi bir sınıfın IDisposable arayüzünü uygulayıp uygulamadığını kontrol edin. Nesnenin semantiği, bir Kapat yönteminin bir Dispose yönteminden daha mantıklı olacağı şekildeyse, Dispose'a ek olarak bir Kapat yöntemi sağlayın".

**İstisna işleme**
Yönetme kodu, diğer nedenlerin yanı sıra güvenlik amacıyla istisna işleme kullanmalıdır. Bu önerilere uyduğunuzdan emin olun:
*Döngülerde istisna işleminden kaçının, gerekliyse try/catch bloğu kullanın.
*İstisnaları yutan kodu tanımlayın
*İstisnaları sadece uygulamadaki akışı kontrol etmek için değil, beklenmedik durumlar için kullanın

## Araçlar

**FxCop**
FxCop, kaynak kodu değil ikili derlemeleri analiz eden bir analiz aracıdır. Araç önceden tanımlanmış bir dizi kurala sahiptir ve bunları yapılandırmak ve genişletmek mümkündür.

Güvenlikle ilgili mevcut kurallardan bazıları şunlardır (CodePlex, 2010):

**Tablo 17:** FxCop

| Kural | Açıklama |
|---|---|
| EnableEventValidationShouldBeTrue | EnableEventValidation yönergesinin belirli bir sayfada devre dışı bırakılıp bırakılmadığını doğrular. |
| ValidateRequestShouldBeEnabled | ValidateRequest yönergesinin belirli bir sayfada devre dışı bırakılıp bırakılmadığını doğrular. |
| ViewStateEncryptionModeShouldBeAlways | ViewStateEncryptionMode yönergesinin belirli bir sayfada Never olarak ayarlanıp ayarlanmadığını doğrular. |
| EnableViewStateMacShouldBeTrue | EnableViewStateMac yönergesinin belirli bir sayfada false olarak ayarlanıp ayarlanmadığını doğrular. |
| EnableViewStateShouldBeTrue | EnableViewState yönergesinin belirli bir sayfada false olarak ayarlanıp ayarlanmadığını doğrular. |
| ViewStateUserKeyShouldBeUsed | CSRF'yi önlemek için Page.ViewStateUserKey'in uygulamada kullanılıp kullanılmadığını doğrular. |
| DebugCompilationMustBeDisabled | Hata ayıklama derlemesinin kapalı olduğunu doğrular. Bu, hata ayıklama kodunun etkinleştirilmesi ve ek kapsamlı hata mesajlarının döndürülmesiyle ilgili olası performans ve güvenlik sorunlarını ortadan kaldırır. |
| CustomErrorPageShouldBeSpecified | CustomErrors bölümünün, hata durumunda kullanımları yeniden yönlendirmek için varsayılan bir URL'ye sahip olacak şekilde yapılandırıldığını doğrular. |
| FormAuthenticationShouldNotContainFormAuthenticationCredentials | Form kimlik doğrulama yapılandırması altında hiçbir kimlik bilgisi belirtilmediğini doğrular. |
| EnableCrossAppRedirectsShouldBeTrue | system.web.authentication.forms enableCrossAppRedirects öğesinin true olarak ayarlandığını doğrular. Ayarlar, kimlik doğrulama işleminden sonra kullanıcının başka bir uygulama url'sine yönlendirilip yönlendirilmeyeceğini belirtir. Ayar false ise, kimlik doğrulama işlemi başka bir uygulamaya veya ana bilgisayara yeniden yönlendirmeye izin vermez. Bu, bir saldırganın kimlik doğrulama işlemi sırasında kullanıcıyı başka bir siteye yönlendirmeye zorlamasını önlemeye yardımcı olur. Bu saldırı genellikle Open redi- rect olarak adlandırılır ve çoğunlukla kimlik avı saldırıları sırasında kullanılır. |
| FormAuthenticationProtectionShouldBeAll | system.web.authentication.forms korumasındaki koruma özniteliğinin, uygulamanın kimlik doğrulama çerezini korumaya yardımcı olmak için hem veri doğrulama hem de şifreleme kullandığını belirten Tümü olarak ayarlandığını doğrular. |
| FormAuthenticationRequireSSLShouldBeTrue | system.web.authentication.forms yapılandırma öğesindeki requireSSL özniteliğinin True olarak ayarlandığını doğrulayarak kimlik doğrulama çerezini güvenli özniteliği belirtmeye zorlar. Bu, tarayıcıyı çerezi yalnızca SSL üzerinden sağlamaya yönlendirir. |
| FormAuthenticationSlidingExpirationShouldBeFalse | Site HTTP üzerinden sunulurken system.web.authentication.forms slidingExpiration öğesinin false olarak ayarlandığını doğrular. Bu, kimlik doğrulama çerezini her istekle yenilenmek yerine sabit bir zaman aşımı değerine sahip olmaya zorlayacaktır. Çerez açık metin ağı üzerinden geçeceğinden ve potansiyel olarak ele geçirilebileceğinden, çerezde sabit bir zaman aşımı değerine sahip olmak, çerezin tekrar oynatılabileceği süreyi sınırlayacaktır. Çerez yalnızca HTTPS üzerinden gönderiliyorsa, ele geçirilme olasılığı daha düşüktür ve slidingExpiration ayarının True olması, zaman aşımının her istekten sonra yenilenmesine neden olarak daha iyi bir kullanıcı deneyimi sağlar. |
| HttpCookiesHttpOnlyCookiesShouldBeTrue | system.web.httpCookies httpOnlyCookies yapılandırma ayarının True olarak ayarlandığını doğrular, bu da tüm çerezlerin HttpOnly özniteliğiyle gönderilmesini zorlar. |
| HttpCookiesRequireSSLShouldBeTrue | system.web.httpCookies requireSSL yapılandırmasının True olarak ayarlandığını doğrular, bu da tüm çerezlerin güvenli öznitelik ile gönderilmesini zorlar. Bu, tarayıcının çerezi yalnızca SSL üzerinden sağlayacağını gösterir. |
| TraceShouldBeDisabled | system.web.trace enabled ayarının izlemeyi devre dışı bırakan false olarak ayarlandığını doğrular. Bir saldırganın uygulamanız hakkında izleme bilgilerinden bilgi edinemeyeceğinden emin olmak için üretim sunucularında izlemeyi devre dışı bırakmanız önerilir. İzleme bilgileri, bir saldırganın uygulamanızı araştırmasına ve tehlikeye atmasına yardımcı olabilir. |

| Kural | Açıklama |
|---|---|
| AnonymousAccessIsEnabled | Yetkilendirme bölümünün anonim erişime izin verip vermediğini görmek için web.config dosyasına bakın. |
| RoleManagerCookieProtectionShouldBeAll | system.web.rolemanager cookieProtection öğesinin, çerezin sunucu tarafından hem şifrelenmesini hem de doğrulanmasını zorunlu kılan Tümü olarak ayarlandığını doğrular. |
| RoleManagerCookieRequireSSLShouldBeTrue | system.web.rolemanager cookieRequireSSL özniteliğinin True olarak ayarlandığını doğrulayarak rol yöneticisi çerezini güvenli özniteliği belirtmeye zorlar. Bu, tarayıcıyı çerezi yalnızca SSL üzerinden sağlamaya yönlendirir. |
| RoleManagerCookieSlidingExpirationShouldBeTrue | Site HTTP üzerinden sunulurken system.web.rolemanager cookieSlidingExpiration öğesinin false olarak ayarlandığını doğrular. Bu, kimlik doğrulama çerezini her istekle yenilenmek yerine sabit bir zaman aşımı değerine sahip olmaya zorlayacaktır. Çerez açık metin ağı üzerinden geçeceğinden ve potansiyel olarak ele geçirilebileceğinden, çerezde sabit bir zaman aşımı değerine sahip olmak, çerezin tekrar oynatılabileceği süreyi sınırlayacaktır. Çerez yalnızca HTTPS üzerinden gönderiliyorsa, ele geçirilme olasılığı daha düşüktür ve slidingExpiration ayarının True olması, zaman aşımının her istekten sonra yenilenmesine neden olarak daha iyi bir kullanıcı deneyimi sağlar. |
| PagesEnableViewStateMacShouldBeTrue | Viewstate mac'in etkin olduğunu doğrular. |
| PagesEnableEventValidationMustBeTrue | Olay doğrulamanın etkin olduğunu doğrular. |
| HttpRuntimeEnableHeaderCheckingShouldBeTrue | system.web.httpRuntime enableHeaderChecking özniteliğinin true olarak ayarlandığını doğrular. Bu ayar, ASP.NET'in istek üstbilgisini olası ekleme saldırılarına karşı denetleyip denetlemeyeceğini belirtir. Bir saldırı algılanırsa, ASP.NET bir hatayla yanıt verir. Bu, ASP.NET'i istemci tarafından gönderilen başlıklara ValidateRequest korumasını uygulamaya zorlar. Bir saldırı algılanırsa uygulama HttpRequestValidationException atar. |
| PagesValidateRequestShouldBeEnabled | validateRequest'in etkin olduğunu doğrulayın. |
| PagesViewStateEncryptionModeShouldBeAlways | Görünüm durumu şifreleme modunun asla şifreleme olarak yapılandırılmadığını doğrular. |
| CustomErrorsModeShouldBeOn | system.web.customErrors modunun On veya RemoteOnly olarak ayarlandığını doğrular. Bu, ASP.NET tarafından uzak kullanıcılara döndürülen kuyruklu hata mesajını devre dışı bırakır. |
| MarkVerbHandlersWithValidateAntiforgeryToken | ValidateAntiforgeryTokenAttribute, ASP.NET MVC uygulamalarına karşı olası CSRF saldırılarına karşı koruma sağlamak için kullanılır. |


A6

# A6 HASSAS VERI MARUZIYETI

Birçok web uygulaması kredi kartları, vergi kimlikleri ve kimlik doğrulama bilgileri gibi hassas verileri düzgün bir şekilde korumaz. Saldırganlar, kredi kartı dolandırıcılığı, kimlik hırsızlığı veya diğer suçları işlemek için bu tür zayıf korunan verileri çalabilir veya değiştirebilir. Hassas veriler, dinlenme veya aktarım sırasında şifreleme gibi ekstra korumanın yanı sıra tarayıcı ile alışveriş yapılırken özel önlemleri hak eder.

## 12.1 Kriptografik Kontroller

Yazılım geliştiricileri, mimarlar ve tasarımcılar, belirli bir uygulamanın hangi kategoride yer alacağına karar vermede ön plandadır. Kriptografi, bekleyen verilerin güvenliğini (şifreleme yoluyla), veri bütünlüğünün uygulanmasını (karma/özetleme yoluyla) ve verilerin inkar edilmemesini (imzalama yoluyla) sağlar. Bu kriptografik kodun verileri yeterince koruduğundan emin olmak için tüm kaynak kodlar güçlü anahtar boyutlarına sahip standart (güvenli) algoritmalar kullanmalıdır.

Kriptografik kodun uygulanmasındaki yaygın kusurlar arasında standart olmayan kriptografik algoritmaların kullanılması, kriptografi (standart ve standart olmayan) algoritmalarının özel olarak uygulanması, kriptografik olarak güvenli olmayan standart algoritmaların kullanılması (örn. DES) ve güvenli olmayan anahtarların uygulanması herhangi bir uygulamanın genel güvenlik duruşunu zayıflatabilir. Yukarıda bahsedilen kusurların uygulanması, saldırganların hassas verilerin şifresini çözmek için kriptanalitik araçlar ve teknikler kullanmasını sağlar.

## 12.2 Açıklama

Birçok şirket, müşterileri için tıbbi bilgiler veya kredi kartı numaraları gibi hassas bilgileri işler ve sektör düzenlemeleri, müşterilerin bilgilerini korumak için bu hassas bilgilerin şifrelenmesi gerektiğini belirtir. Tıp sektöründe HIPAA yönetmelikleri işletmelere tıbbi verilere hangi korumaların uygulanması gerektiğini bildirir, finans sektöründe ise birçok yönetmelik PII (kişisel olarak tanımlanabilir bilgiler) kontrollerini kapsar.

Düzenleyici cezaların mali etkisi ne olursa olsun, bir uygulama tarafından işlenen bilgilerin korunması (şifreleme veya hash yoluyla) için gizlilik ve dolandırıcılık tespiti/korunması dahil olmak üzere birçok iş nedeni vardır. Uygulamanın işlediği tüm hassas veriler tanımlanmalı ve şifreleme uygulanmalıdır. Aynı şekilde, hassas verilerin aktarılırken (yani bir bilgisayardan diğerine gönderilirken) ve/veya dururken (yani bir DB'de, dosyada, anahtar zincirinde vb. saklanırken) şifrelenmesi gerekip gerekmediğine karar verilmelidir:

**1)** Taşıma sırasında koruma; bu tipik olarak HTTP protokolü üzerinde seyahat eden verileri şifrelemek için SSL/TLS katmanının kullanılması anlamına gelir, ancak FTPS ve hatta TCP üzerinde SSL de içerebilir. IIS ve Apache Struts gibi çerçeveler SSL/TLS işlevselliği ile birlikte gelir ve bu nedenle geliştirici gerçek TLS şifrelemesini kodlamayacak, bunun yerine TLS güvenliği sağlamak için bir çerçeve yapılandıracaktır.

Ancak burada alınan kararların, mimari düzeyde bile olsa, iyi bilgilendirilmiş olması gerekir ve TLS tasarım kararlarına ilişkin bir tartışma bölüm 1.3'te ele alınmaktadır.

**2)** Beklemede koruma; bu, veritabanındaki kredi kartlarının şifrelenmesini, şifrelerin hashlenmesini, bir mesajın bilgisayarlar arasında değiştirilmediğinden emin olmak için mesaj kimlik doğrulama kodları (MAC'ler) üretilmesini içerebilir. TLS kodunun bir çerçeve ile birlikte geleceği durumlarda, depolanacak verileri şifreleyecek veya hashleyecek kodun tipik olarak kriptografik kütüphaneler tarafından sağlanan API'leri kullanması gerekecektir.

Geliştirici AES algoritmasını uygulamak için kod yazmayacaktır (OpenSSL veya CryptoAPI bunu yapacaktır), geliştirici bir AES implantasyonunu doğru şekilde kullanmak için modüller yazacaktır. Yine güncel algoritmalar,

anahtar depolama ve bölüm 1.4'te ele alınan diğer tasarım kararları ile ilgili doğru kararların alınması gerekir.

**Kriptografi Tanımları**
Trafik ve verilerin şifrelenmesine ilişkin tartışmalara geçmeden önce, kriptografi alanında kullanılan bazı terminoloji **tablo 18**'de tanımlanmıştır.

**Tablo 18:** Kriptografik Tanımlar

| Dönem | Açıklama |
|---|---|
| Kodlama | Verinin bir formdan diğerine dönüştürülmesi, tipik olarak veriyle çalışmayı kolaylaştırmak amacıyla. Örneğin, ikili verilerin (ekrana yazdırılamayan) kopyalanıp yapıştırılabilen yazdırılabilir ASCII formatına kodlanması. Kodlamanın veriyi gizlemeyi amaçlamadığını, kodlanmış veriyi orijinal formuna geri döndürme yönteminin kamuya açık olacağını unutmayın. |
| Entropi | Esasen bu rastgeleliktir. Kriptografik işlevlerin, kaynak verilerin bir saldırganın gerekli anahtar olmadan şifrelemeyi tersine çeviremeyeceği şekilde şifrelenmesine izin vermek için bir tür rastlantısallıkla çalışması gerekecektir. İyi bir entropi kaynağına sahip olmak her kriptografik algoritma için esastır. |
| Hashing | Verilerin 'parmak izi' veya 'hashvalue' olarak adlandırılan geri döndürülemez dönüşümü. Herhangi bir boyutta girdi alınabilir ve her zaman aynı boyutta çıktı (algoritma için) ile sonuçlanır. Amaç, parmak izini daha sonra tekrar kaynak veriye dönüştürmek değil, aynı parmak izini üretip üretmediklerini belirlemek için karma algoritmayı iki veri kümesi üzerinde çalıştırmaktır. Bu, verilerin tahrif edilmediğini gösterecektir. |
| Tuz | Parmak izi sonucunu değiştirmek için bir hashing algoritmasına eklenebilen gizli olmayan bir değer. Hashing algoritmalarına karşı yapılan saldırılardan biri, tüm kaynak değerlerinin önceden hesaplandığı ve bir tablo oluşturulduğu 'gökkuşağı tablosu saldırısı'dır. Saldırgan daha sonra bir parmak izi alabilir, tabloda arayabilir ve orijinal veriye karşılık getirebilir. Hash edilecek her veri için benzersiz bir tuz değeri kullanmak gökkuşağı tablolarına karşı koruma sağlar, çünkü her tuz değeri için bir gökkuşağı tablosu oluşturulması gerekir ve bu da saldırganın harcayacağı zamanı büyük ölçüde uzatır. Tuz bir sır değildir ve parmak iziyle birlikte saklanabilir veya gönderilebilir. |
| Şifreleme | Kaynak verilerin orijinal kaynağa geri döndürülebilecek şifrelenmiş bir forma dönüştürülmesi. Tipik olarak şifrelemek için kullanılan algoritmalar kamuya açıktır, ancak dönüşümü yönlendirmek için gizli bir 'anahtara' dayanır. Anahtara sahip olmayan bir saldırganın verileri orijinal kaynağa geri dönüştürememesi gerekir. |
| Simetrik Şifreleme | Aynı anahtarın hem gönderici hem de alıcı tarafından bilindiği bir şifreleme biçimi. Bu hızlı bir şifreleme şeklidir, ancak simetrik anahtarı gönderici ve alıcı arasında iletmek için güvenli, bant dışı bir yöntem gerektirir. |
| Açık Anahtar Şifreleme (PKI) | Biri veriyi şifrelemek, diğeri de veriyi orijinal haline geri döndürmek için olmak üzere iki anahtar kullanan bir şifreleme biçimi. Bu daha yavaş bir şifreleme yöntemidir ancak anahtarlardan biri herkes tarafından bilinebilir ('açık anahtar' olarak adlandırılır). Diğer anahtar 'özel anahtar' olarak adlandırılır ve gizli tutulur. Açık anahtarla şifrelenen herhangi bir veri, özel anahtar kullanılarak orijinal haline geri döndürülebilir. Benzer şekilde, özel anahtarla şifrelenen herhangi bir veri, açık anahtar kullanılarak orijinal haline geri döndürülebilir. |
| Sertifika | Bir varlık (örn. kişi, şirket) ile bir açık anahtar arasındaki ilişki. Tipik olarak bu, belirli güvenilir varlıkların (örneğin internet TLS'deki Sertifika Yetkilileri) bir hak sahibinin kimlik bilgilerinin doğrulamasını yaptığı ve beyan edilen bir açık anahtarın varlığa ait olduğunu (kendi sertifikalarını kullanarak) iddia ettiği bir açık anahtar altyapısının bir parçasını oluşturur. |

## 12.3 Ne İncelenmeli? Taşıma Sırasında Koruma

Secure Socket Layer (SSL) ve Transport Layer Security (TLS) terimleri genellikle birbirlerinin yerine kullanılır. Aslında SSL v3.1, TLS v1.0 ile eşdeğerdir. Ancak, SSL ve TLS'nin farklı sürümleri modern web tarayıcıları ve çoğu modern web çerçevesi ve platformu tarafından desteklenmektedir. SSL protokolüne yönelik saldırılardaki gelişmeler protokolün saldırılara karşı daha zayıf olduğunu gösterdiğinden, bu kılavuzda HTTP veya TCP protokolleri üzerinden aktarım katmanı güvenliğini ifade etmek için TLS terimi kullanılacaktır.

Aktarım katmanı güvenliğinin birincil faydası, web uygulama verilerinin istemciler (web tarayıcıları) ile web uygulama sunucusu arasında ve web uygulama sunucusu ile arka uç ve tarayıcı tabanlı olmayan diğer kurumsal bileşenler arasında iletilirken yetkisiz ifşa ve değişikliklere karşı korunmasıdır.
Teorik olarak, bilgisayarlar arası iletişimi korumak için TLS kullanma kararı, trafiğin doğasına veya arayüz üzerinde mevcut olan işlevselliğe dayanmalıdır. Eğer hassas bilgiler arayüz üzerinden geçiyorsa

arayüzünde TLS, gizli dinleyicilerin verileri görüntülemesini veya değiştirmesini engelleyecektir. Aynı şekilde arayüz para transferine ya da hassas işlevlerin başlatılmasına izin veriyorsa, TLS kullanıcıya bu işlevleri gerçekleştirme yetkisi veren ilgili oturum açma ya da oturum bilgilerini koruyacaktır. Bununla birlikte, sertifika fiyatlarının düşmesi ve çerçevelerdeki TLS yapılandırmasının kolaylaşmasıyla, bir arayüzün TLS koruması büyük bir çaba değildir ve birçok web sitesi tüm siteleri için TLS korumaları kullanmaktadır (yani, yalnızca HTTPS sayfaları vardır, HTTP sayfaları mevcut değildir).

TLS'nin sunucu doğrulama bileşeni, sunucunun istemciye kimlik doğrulamasını sağlar. İstemci tarafı sertifikaları gerektirecek şekilde yapılandırılırsa TLS, istemcinin sunucuya kimlik doğrulamasında da rol oynayabilir. Ancak pratikte istemci tarafı sertifikaları, istemciler için kullanıcı adı ve parola tabanlı kimlik doğrulama modellerinin yerine sıklıkla kullanılmaz.

**Onaylanmış Uygulamaları Kullanma**
ABD hükümeti, TLS'de kullanılanlar da dahil olmak üzere çeşitli kriptografik işlevlerin güçlü ve güvenli bir şekilde uygulanmasını sağlamak için onaylanmış bir yazılım listesi sağlar. Bu liste FIPS 140-2 onaylı kriptomodüller olarak adlandırılır.

Bir kriptomodül, ister bir yazılım kütüphanesi ister bir donanım cihazı olsun, kriptografik algoritmaları (simetrik ve asimetrik algoritmalar, hash algoritmaları, rastgele sayı üreteci algoritmaları ve mesaj kimlik doğrulama kodu algoritmaları) uygular. Bir kriptomodülün ve hizmetlerinin (ve kriptomodülü çağıran web uygulamalarının) güvenliği, bu üç parçanın her birinin doğru şekilde uygulanmasına ve entegrasyonuna bağlıdır. Buna ek olarak, kriptomodül güvenli bir şekilde kullanılmalı ve erişilmelidir. TLS'nin faydalarından yararlanmak için FIPS 140-2 onaylı bir TLS hizmeti (örn. kütüphane, web çerçevesi, web uygulama sunucusu) kullanmak önemlidir. Buna ek olarak, FIPS 140-2 onaylı kriptomodülün beklenen güvenlik hizmetlerini beklenen şekilde sağladığına dair yüksek derecede kesinlik sağlamak için kriptomodülün onaylı veya izin verilen bir modda kurulması, yapılandırılması ve çalıştırılması gerekir.

**TLS şifrelemesini işleyen tasarımları veya kodları incelerken dikkat edilmesi gereken öğeler şunlardır:**
• Oturum açma sayfaları ve kimliği doğrulanmış tüm sayfalar için TLS kullanın. Oturum açma açılış sayfası için TLS kullanılmaması, bir saldırganın oturum açma formu eylemini değiştirmesine ve kullanıcının kimlik bilgilerinin rastgele bir konuma gönderilmesine neden olur. Oturum açma işleminden sonra kimliği doğrulanmış sayfalar için TLS kullanılmaması, bir saldırganın şifrelenmemiş oturum kimliğini görüntülemesine ve kullanıcının kimliği doğrulanmış oturumunu tehlikeye atmasına olanak tanır.

• Hassas verileri iletirken veya kimliği doğrulanmış işlevleri açığa çıkarırken TLS'yi dahili olarak kullanın. Hassas verileri ileten hem harici hem de dahili tüm ağlar TLS veya eşdeğer bir aktarım katmanı güvenlik mekanizması kullanmalıdır. Dahili ağa erişimin "çalışanlarla sınırlı" olduğunu iddia etmek yeterli değildir. Yakın zamanda yaşanan çok sayıda veri ihlali, dahili ağın saldırganlar tarafından ihlal edilebileceğini göstermiştir. Bu saldırılarda, iç ağda gönderilen şifrelenmemiş hassas verilere erişmek için koklayıcılar kurulmuştur.

• Tüm arayüzlerin (veya sayfaların) yalnızca HTTPS üzerinden erişilebilir olmasını tercih edin. TLS üzerinden erişilebilen tüm sayfalar, TLS olmayan bir bağlantı üzerinden erişilememelidir. Bir kullanıcı, uygulamanın kimliği doğrulanmış bölümünde yanlışlıkla bir HTTP sayfasına (örn. http://example.com/myaccount) yer işareti koyabilir veya manuel olarak bir URL yazabilir.

• Kimlik doğrulama çerezleri için "secure" ve "http-only" çerez bayraklarını kullanın. "Güvenli" bayrağının kullanılmaması, bir saldırganın kullanıcının tarayıcısını sitedeki şifrelenmemiş bir sayfaya istek göndermesi için kandırarak oturum çerezine erişmesini sağlar. "http-only" bayrağı JavaScript işlevlerinin çerez içeriğine erişimini engeller.

- URL'ye hassas veriler koymayın. TLS, taşınırken URL de dahil olmak üzere kablo üzerindeki trafiğin içeriğini koruyacaktır, ancak URL'lerin tarayıcı geçmişi ayarlarında görülebileceğini ve tipik olarak yazıldığını unutmayın

sunucu günlüklerine.

• Hassas verilerin önbelleğe alınmasını önleyin. TLS protokolü yalnızca aktarım halindeki veriler için gizlilik sağlar, ancak istemci veya aracı proxy'lerdeki olası veri sızıntısı sorunlarına yardımcı olmaz.

• Yüksek riskli arayüzler için HTTP Strict Transport Security (HSTS) kullanın. HSTS, herhangi bir web istemcisinin web sitenize TLS olmayan bir protokol üzerinden bağlanmaya çalışmasını engelleyecektir. Sunucu tarafı açısından bakıldığında, eğer TLS olmayan sayfalar sunulmuyorsa bu önemsiz görünebilir, ancak HSTS'yi kuran bir web sitesi istemcileri diğer saldırılardan (örneğin DNS önbellek zehirlenmesi) korur.

• 2048 anahtar uzunluğu (ve üzeri) ve SHA-256 (ve üzeri) kullanın. Şifre anahtarını oluşturmak için kullanılan özel anahtar, özel anahtarın ve ilgili sertifikanın beklenen kullanım ömrü için yeterince güçlü olmalıdır. Mevcut en iyi uygulama en az 2048 bitlik bir anahtar boyutu seçmektir. SHA-1'e yönelik saldırıların zayıflık gösterdiğini ve mevcut en iyi uygulamanın en az SHA-256 veya eşdeğerini kullanmak olduğunu unutmayın.

• Sertifikalarınızda yalnızca belirtilen, tam nitelikli alan adlarını kullanın. Joker karakter sertifikaları veya RFC 1918 adresleri (örn. 10.* veya 192.168.*) kullanmayın. Birden fazla alan adını desteklemeniz gerekiyorsa, sertifikanın geçerli olduğu birden fazla adın belirli bir listesini sağlayan Konu Alternatif Adlarını (SAN'lar) kullanın. Örneğin sertifika, öznenin CN'sini example.com olarak listeleyebilir ve iki SAN listeleyebilir: abc.example.com ve xyz.example. com. Bu sertifikalar bazen "çoklu alan adı sertifikaları" olarak adlandırılır.

• Her zaman zincirdeki tüm sertifikaları sağlayın. Bir kullanıcı bir sunucu veya ana bilgisayarın sertifikasını aldığında, sertifikanın güvenilir bir kök sertifika yetkilisine kadar doğrulanması gerekir. Bu, yol doğrulaması olarak bilinir. Son varlık (sunucu veya ana bilgisayar) sertifikası ile kök sertifika arasında bir veya daha fazla ara sertifika o l a b i l i r . Her iki uç noktayı doğrulamaya ek olarak, istemci yazılımının da tüm ara sertifikaları doğrulaması gerekecektir, bu da istemcinin sertifikalara sahip olmaması durumunda arızalara neden olabilir. Bu durum birçok mobil platformda meydana gelmektedir.

### 12.4 Ne İncelenmeli? Dinlenme Sırasında Koruma

Genel bir öneri olarak, şirketler kendi özel kriptografik kütüphanelerini ve algoritmalarını oluşturmamalıdır. Kriptografik algoritmalar geliştiren gruplar, kuruluşlar ve bireyler ile kriptografiyi yazılımda ya da donanımda uygulayanlar arasında büyük bir fark vardır. Uzmanlar tarafından geliştirilmiş ve endüstri tarafından test edilmiş yerleşik bir kriptografik kütüphane kullanmak, bir şirketin koduna kriptografik işlevler eklemenin en güvenli yoludur. Çeşitli dillerde ve ortamlarda kullanılan bazı yaygın kütüphane örnekleri aşağıdaki tabloda yer almaktadır.

**Tablo 19:** Ortama göre popüler kriptografik uygulamalar

| Dil | Kütüphaneler | Tartışma |
|---|---|---|
| C# .NET | 'System.Security' içindeki sınıf kütüphaneleri. Kriptography' | C#.NET'te kodlanan uygulamalar için 'System.Security.Cryptog- raphy' içinde kullanılması gereken sınıf kütüphaneleri ve uygulamalar vardır. .NET içindeki bu isim alanı, kullanmak için kriptografi konusunda yetkin bilgi gerektirmeyen bir dizi çözgü sağlamayı amaçlamaktadır. |
| C/C++ (Win32) | CryptoAPI ve DPAPI | Win32 platformlarında çalışan C/C++ kodu için CreyptoAPI ve DPAPI önerilir. |
| C/C++ (Linux) | OpenSSL, NSS, boringssl | Linux/Unix işletim sistemlerinde C/C++ için OpenSSL, NSS veya bu kütüphanelerin birçok çatalından birini kullanın. |

| | | |
|---|---|---|
| ASP | CryptoAPI ve DPAPI | Classis ASP sayfalarının kriptografik işlevlere doğrudan erişimi yoktur, bu nedenle tek yol Visual C++ veya Visual Basic'te CryptoAPI veya DPAPI çağrılarını uygulayan COM sarmalayıcıları oluşturmaktır. Daha sonra bunları ASP sayfalarından Server. CreateObject yöntemini kullanarak çağırın. |
| Java | Java Cryptography Extension, BouncyCastle, Bahar Güvenlik | JCE, herhangi bir kriptografik kütüphanenin geliştiriciye kriptografik işlevler sağlamak için uygulayabileceği standart bir API'dir. Oracle, Kriptografik Hizmet Sağlayıcıları olarak hareket eden ve/veya JCE'n i n temiz oda uygulamalarını sunan şirketlerin bir listesini sağlar. BouncyCastle en popüler uygulamalardan biridir. Spring Secuirty, Spring'in halihazırda kullanıldığı uygulamalarda da popülerdir. |

• P122 - biraz yanlış nokta girintileri - bir aramada hızlıca.

Kaynak kodu içinde sağlam şifreleme mekanizmaları uygulamanın güvenli bir yolu, Microsoft Veri Koruma API'si (DPAPI) [4] veya Java Kriptografi Uzantısı (JCE) [5] kullanılarak FIPS [7] uyumlu algo- ritmalar kullanmaktır.

**Bir şirket kriptografik kod stratejinizi oluştururken aşağıdakiler için minimum standartları belirlemelidir:**

- Uygulamalar tarafından hangi standart algoritmaların kullanılacağı
- Desteklenecek minimum anahtar boyutları
- Ne tür veriler şifrelenmelidir

**Kriptografiyi ele alan kodu gözden geçirirken şunlara dikkat edin:**

- Yeterince güçlü şifreleme algoritmaları kullanılıyor mu ve bu algoritmaların uygulanması FIPS-140 standartlarına uygun mu?
- Doğru türde kriptografik algoritma kullanılıyor mu, simetrik anahtarla şifrelenmesi gereken veriler hash ediliyor mu? Simetrik anahtarı karşı tarafa güvenli bir şekilde aktarmanın bir yolu yoksa, açık anahtarlı kriptografik algoritmalar kullanılıyor mu?
- Herhangi bir kriptografik sistemde anahtarın korunması en önemli husustur. Simetrik veya özel anahtarın açığa çıkması, şifrelenmiş verilerin artık gizli olmadığı anlamına gelir. Anahtarları kimin girebileceğini veya görüntüleyebileceğini ve anahtarların uygulamalar içinde nasıl kullanılacağını sıkı bir şekilde kontrol edin.
- Kriptografik süreçleri ve algoritmaları uygulayan tüm kodlar bir dizi şirket veya düzenleyici şartnameye göre gözden geçirilmeli ve denetlenmelidir. Bir kuruluşun 'güçlü şifrelemeyi' ne olarak gördüğü konusunda üst düzey kararlar alınmalı (ve sürekli olarak yeniden gözden geçirilmelidir) ve tüm uygulama örnekleri bu standarda uygun olmalıdır.
- Kriptografik modüller çok iş parçacıklı uygulamalarla yüksek yük altında test edilmeli ve şifrelenmiş her bir veri parçasının doğru şekilde şifrelendiğinden ve şifresinin çözüldüğünden emin olmak için kontrol edilmelidir.
- Net'te kriptografi örnekleri için MSDN Kütüphanesi Güvenlik Uygulamaları: .NET Framework 2.0 Bir Bakışta Güvenlik Uygulamaları

  - Veri Koruma API'sinin (DPAPI) kullanılıp kullanılmadığını kontrol edin.
  - Tescilli algoritmaların kullanılmadığını doğrulayın.
  - PRNG için RNGCryptoServiceProvider'ın kullanıldığını kontrol edin.
  - Anahtar uzunluğunun en az 128 bit olduğunu doğrulayın.
- ASP'de, ASP'nin kriptografik işlevlere doğrudan erişimi olmadığından, tüm bu kontrolleri COM sarmalayıcı üzerinde gerçekleştirin

  - COM nesnesinde Veri Koruma API'sinin (DPAPI) veya CryptoAPI'nin kullanıldığını kontrol edin

sayfanın altında
mermi açısından
tekrar ele alalım

- Tescilli algoritmaların kullanılmadığını doğrulayın
- PRNG için RNGCryptoServiceProvider'ın kullanıldığını kontrol edin
- Anahtar uzunluğunun en az 128 bit olduğunu doğrulayın

• Java için Java Kriptografi Uzantısının (JCE) kullanıldığını kontrol edin

- Tescilli algoritmaların kullanılmadığını doğrulayın
- PRNG için SecureRandom (veya benzeri) kullanıldığını kontrol edin
- Anahtar uzunluğunun en az 128 bit olduğunu doğrulayın

### Kötü Uygulama: Güvensiz Kriptografik Algoritmaların Kullanımı

DES ve SHA-0 algoritmaları kriptografik olarak güvensizdir. **Örnek 12.1**'deki örnek, kullanılmaması gereken DES (Java Kriptografik Uzantıları kullanılarak kullanılabilir) kullanan bir kriptografik modülü özetlemektedir. Ayrıca, SHA-1 ve MD5 ileriye dönük yeni uygulamalarda kullanılmamalıdır.

**Örnek 12.1**

```
paket org.badexample.crypto;

<snip>
try {
                    /** Adım 1. KeyGenerator kullanarak bir DES anahtarı
                    oluşturun */ KeyGenerator keyGen =
                    KeyGenerator.getInstance("DES"); SecretKey secretKey =
                    keyGen.generateKey();

                    /** Adım2. Aşağıdaki parametreleri belirterek bir Şifre oluşturun
                    *                    a. Algoritma adı - işte DES
                    *                    b. Mod - burada CBC
                    *                    c. Dolgu - PKCS5Padding */
                    Cipher desCipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
<snip>
```

### İyi Uygulama: Güçlü Entropi Kullanın

**Örnek 12.2**'deki kaynak kodu, güçlü entropi kullanımına göre güvenli anahtar üretimini özetlemektedir:

**Örnek 12.2**

```
package org.owasp.java.crypto;
import java.security.SecureRandom;
import java.security.NoSuchAlgorithmException;
import sun.misc.BASE64Encoder;
/**
 * Bu program Güvenli Rastgele Sayı Oluşturma işlevselliği sağlar.
 * SecureRandom aracılığıyla bir Rastgele sayı oluşturmanın 2 yolu vardır.
 * 1. Rastgele Baytlar oluşturmak için nextBytes yöntemini çağırarak
```

```
 * 2. Bir Random nesnesini yeniden tohumlamak için setSeed(byte[]) kullanma
*/
public class SecureRandomGen {
        public static void main(String[] args) {
                try {
            // Güvenli bir rastgele sayı üreteci başlatın
            SecureRandom secureRandom = SecureRandom.getInstance("SHA512");

            // Yöntem 1 - Rastgele Baytlar oluşturmak için nextBytes yöntemini
            çağırma byte[] bytes = new byte[512];
            secureRandom.nextBytes(bytes);

            // secureRandom.nextDouble() çağrısı yapılarak SecureRandom sayısının yazdırılması
            System.out.println(" nextBytes() çağrısı yapılarak oluşturulan Secure Random # is " + secureRandom.
nextDouble());

            // Yöntem 2 - Bir Random nesnesini yeniden tohumlamak için
            setSeed(byte[]) kullanımı int seedByteCount = 10;
            byte[] seed = secureRandom.generateSeed(seedByteCount);

            secureRandom.setSeed(seed);
            System.out.println(" setSeed(byte[]) kullanılarak oluşturulan Secure Random # is " + secureRandom.
nextDouble());

          } catch (NoSuchAlgorithmException noSuchAlgo)
                {
                        System.out.println(" Böyle Bir Algoritma Yok" + noSuchAlgo);
                }
        }
}
```

### İyi Uygulama: Güçlü Algoritmalar Kullanın

Aşağıda AES uygulaması gösterilmektedir (Java Kriptografik Uzantılarını Kullanma başına kullanılabilir):

**Örnek 12.3**

```
paket org.owasp.java.crypto;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.Cipher;

import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
import java.security.InvalidAlgorithmParameterException;
import javax.crypto.NoSuchPaddingException;
```

```
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;

import sun.misc.BASE64Encoder;

/**
 * Bu program aşağıdaki kriptografik işlevleri sağlar
 * 1. AES kullanarak şifreleme
 * 2. AES kullanarak şifre çözme
 *
 * Yüksek Seviye Algoritma :
 * 1. Bir DES anahtarı oluşturun (bu aşamada Anahtar boyutunu belirtin)
 * 2. Şifreyi Oluşturun
 * 3. Şifrelemek için : Şifreleme için Şifreyi Başlatın
 * 4. Şifre Çözmek İçin : Şifre Çözme için Şifreyi Başlatın
 */

public class AES {
        public static void main(String[] args) {

                String strDataToEncrypt = new String();
                String strCipherText = new String();
                String strDecryptedText = new String();
try{
                /**
                 *  Adım 1. KeyGenerator kullanarak bir AES anahtarı oluşturun
                 *                  Anahtar boyutunu 128 olarak başlatın
                 */
                KeyGenerator keyGen = KeyGenerator.getInstance("AES");
                keyGen.init(128);
                SecretKey secretKey = keyGen.generateKey();

                /**
                 *  Adım 2. Aşağıdaki parametreleri belirleyerek bir Şifre oluşturun
                 *                      a. Algoritma adı - burada AES
                 */
                Cipher aesCipher = Cipher.getInstance("AES");

                /**
                 *  3. Adım Şifreleme için Şifreyi Başlatın
                 */
                aesCipher.init(Cipher.ENCRYPT_MODE,secretKey);

                /**
                 *  4. Adım Verileri Şifreleyin
                 *                  1. Verileri Bildirin / Başlatın. Burada veri String tipindedir
                 *                  2. Girilen Metni Bayta Dönüştürme
```

```
                *                       3. doFinal yöntemini kullanarak baytları şifreleyin
                */
                strDataToEncrypt ="Hello World of Encryption using
                AES"; byte[] byteDataToEncrypt =
                strDataToEncrypt.getBytes();
                byte[] byteCipherText = aesCipher.doFinal(byteDataToEncrypt);
                strCipherText = new BASE64Encoder().encode(byteCipherText);
                System.out.println("AES kullanılarak oluşturulan Şifreli Metin "
                +strCipherText);

                /**
                */ Adım 5. Verilerin Şifresini Çözme
                *                       1. Şifre Çözme için Şifreyi Başlatma
                *                       2. doFinal yöntemini kullanarak şifre baytlarının şifresini çözün
*/
                aesCipher.init(Cipher.DECRYPT_MODE,secretKey,aesCipher.getParameters());
                byte[] byteDecryptedText = aesCipher.doFinal(byteCipherText);
                strDecryptedText = new String(byteDecryptedText);
                System.out.println(" Şifresi Çözülen Metin mesajı " +strDecryptedText);
                }

                catch (NoSuchAlgorithmException noSuchAlgo)
                {
                        System.out.println(" Böyle Bir Algoritma Yok" + noSuchAlgo);
                }
                        catch (NoSuchPaddingException noSuchPad)
                        {
                                System.out.println(" Böyle Bir Dolgu Yok" + noSuchPad);
                        }
                                catch (InvalidKeyException invalidKey)
                                {
                                        System.out.println(" Geçersiz Anahtar" + invalidKey);
                                }
                                catch (BadPaddingException badPadding)
                                {
                                        System.out.println(" Bad Padding" + badPadding);
                                }
                                catch (IllegalBlockSizeException illegalBlockSize)
                                {
                                        System.out.println(" Yasadışı Blok Boyutu" + illegalBlockSize);
                                }
                                catch (InvalidAlgorithmParameterException invalidParam)
                                {
                                        System.out.println(" Geçersiz Parametre" + invalidParam);
                                }
        }
}
```

### 1.1.4 Referanslar

**[1]** Bruce Schneier, Uygulamalı Kriptografi, John Wiley & Sons, 2. baskı, 1996.

**[2]** Michael Howard, Steve Lipner, The Security Development Lifecycle, 2006, s. 251 - 258

**[3]** NET Framework Geliştirici Kılavuzu, Kriptografik Hizmetler, **http://msdn2.microsoft.com/en-us/library/93bsk-f9z.aspx**

**[4]** Microsoft Geliştirici Ağı, Windows Veri Koruma, **http://msdn2.microsoft.com/en-us/library/ms995355.aspx**

**[5]** Sun Geliştirici Ağı, Java Kriptografi Uzantısı, **http://java.sun.com/products/jce/**

**[6]** Sun Developer Network, Kriptografik Hizmet Sağlayıcıları ve Temiz Oda Uygulamaları, **http://java.sun.com/products/jce/jce122_providers.html**

**[7]** Federal Bilgi İşlem Standartları, **http://csrc.nist.gov/publications/fips/**

### 12.5 Şifreleme, Hashing ve Tuzlama

Hash fonksiyonu olarak da adlandırılan kriptografik hash algoritması, rastgele bir veri bloğundan (ikili veri dizisi) rastgele bir eşleme sağlamak ve "mesaj özeti" olarak bilinen sabit boyutlu bir bit dizisi döndürmek ve belirli bir güvenlik elde etmek için tasarlanmış bir bilgisayar algoritmasıdır.

Kriptografik karma işlevleri dijital imzalar, mesaj kimlik doğrulama kodları (MAC'ler), diğer kimlik doğrulama biçimleri ve bilgi altyapısındaki diğer birçok güvenlik uygulamasını oluşturmak için kullanılır. Ayrıca kullanıcı parolalarını açık metin olarak saklamak yerine veritabanlarında saklamak ve web uygulamaları için oturum yönetiminde veri sızıntısını önlemeye yardımcı olmak için kullanılırlar. Bir kriptoloji işlevi oluşturmak için kullanılan asıl algoritma uygulamaya göre değişir (SHA-256, SHA-512, vb.)

Programcı tarafından hashing için oluşturulan bir algoritmayı asla bir kod incelemesinde kabul etmeyin. Her zaman dil, çerçeve veya ortak (güvenilir) kriptografik kütüphaneler tarafından sağlanan kriptografik işlevleri kullanın. Bu işlevler deneyimli kriptograflar tarafından iyi incelenmiş ve test edilmiştir.
Amerika Birleşik Devletleri'nde 2000 yılında Ticaret Bakanlığı İhracat Bürosu şifreleme ihracatı yönetmeliklerini revize etmiştir. Yeni ihracat düzenlemelerinin sonuçları, düzenlemelerin büyük ölçüde gevşetilmiş olmasıdır. Ancak kod kaynak ülke dışına ihraç edilecekse, ihracat ve ithalat ülkeleri için mevcut ihracat yasaları uygunluk açısından gözden geçirilmelidir.

Örneğin, mesajın dijital imzası yerine mesajın tamamı hash edilirse, Ulusal Güvenlik Ajansı (NSA) bunu yarı-şifreleme olarak değerlendirir ve Devlet kontrolleri uygulanır.

Kod incelemesi yasal uyumluluğu sağlamak için yapılıyorsa, kuruluş içinde yasal tavsiye almak her zaman geçerli bir seçimdir.

Güvenlik söz konusu olduğunda hiçbir şey sonsuza kadar güvenli değildir. Bu durum özellikle kriptografik hash fonksiyonları için geçerlidir. Windows LanMan hashleri gibi bazı hash algoritmalarının tamamen bozuk olduğu düşünülmektedir. Geçmişte parola karması kullanımı için güvenli kabul edilen MD5 gibi diğerlerinin çarpışma saldırıları gibi bilinen sorunları vardır (çarpışma saldırılarının parola karmalarını etkilemediğini unutmayın). Kodu gözden geçiren kişinin, eski hash fonksiyonlarının zayıflıklarının yanı sıra kriptografik algoritmaların seçimi için mevcut en iyi uygulamaları da anlaması gerekir.

### Tuzlarla Çalışma

Hashing ile ilgili en yaygın programatik sorun şudur:

- Tuz değeri kullanmamak
- Tuz değeri çok kısa olan bir tuz kullanmak
- Aynı tuz değeri birden fazla karmada kullanılır.

Bir tuzun amacı, bir saldırganın önceden hesaplanmış hash saldırısı gerçekleştirmesini zorlaştırmaktır (örneğin, gökkuşağı tabloları kullanarak). Örneğin, SHA512'nin 'password' değerinin **tablo 20'nin** 1. satırında gösterildiği gibi olduğunu ve gökkuşağı tablosuna sahip herhangi bir saldırganın 'password' değerine karşılık gelen hash değerini tespit edeceğini varsayalım. Bir gökkuşağı tablosunu yaklaşık 8 veya 10 karaktere kadar olan değerler için hesaplamanın günler veya haftalar aldığı düşünüldüğünde, bir uygulama herhangi bir tuz kullanmıyorsa bu tabloyu üretmek için harcanan çabaya değer.

Şimdi bir uygulamanın girilen tüm parolalara 'WindowCleaner' tuzunu eklediği bir senaryoyu ele alalım. Artık 'password' hash'i, **tablo 20'nin** 2. satırında gösterilen 'passwordWindowCleaner' hash'ine dönüşür. Bunun saldırganın gökkuşağı tablosunda olması pek olası değildir, ancak saldırgan şimdi (örneğin) sonraki 10 gününü her 8 ila 10 karakter dizisinin sonunda 'WindowCleaner' bulunan yeni bir gökkuşağı tablosu hesaplayarak geçirebilir ve artık hashlenmiş parola veritabanımızın şifresini çözebilir.

Son adımda, bir uygulama her giriş için rastgele bir tuz oluşturabilir ve bu tuzu hash edilmiş parola ile birlikte DB'de saklayabilir. Şimdi kullanıcı1 için rastgele tuz 'a0w8hsdfas8ls587uas87' olsun, yani hashlenecek parola 'passworda0w8hsdfas8ls587uas87' olsun, **tablo 20'nin** 3. satırında gösterilir ve kullanıcı2 için rastgele tuz '8ash87123klnf9d8dq3w' olsun, yani hashlenecek parola 'password8ash87123klnf9d8dq3w' olsun, tablo X'in 4. satırında gösterilir ve tüm kullanıcılar için tekrarlanır.

Artık bir saldırganın şifresini çözmek istediği her kullanıcı için bir gökkuşağı tablosuna ihtiyacı olacaktır - daha önce aynı tuzu kullanarak tüm DB şifrelerinin şifresini çözmek 10 gün sürerken, şimdi kullanıcı1'in şifresi için bir gökkuşağı tablosu oluşturmak 10 gün, kullanıcı2'nin şifresi için bir başka 10 gün vb. sürmektedir. Eğer 100.000 kullanıcı varsa, tüm kullanıcılar için gökkuşağı tabloları oluşturmak 100.000 x 10 gün = 1.000.000 gün veya 2738 yıl sürer.

Görüldüğü gibi, tuzun gizli olması gerekmez, çünkü bir saldırganı yavaşlatan her kullanıcı için benzersiz tuzların kullanılmasıdır.

**Tablo 20:** Tuz kullanımları ve ilişkili parmak izleri

| 'Parola' hash'i için yöntem | Parmak izi |
|---|---|
| Tuz yok | B109F3BBBC244EB82441917ED06D618B9008DD09B3<br>BEFD1B5E07394C706A8BB980B1D7785E5976EC049B<br>46DF5F1326AF5A2EA6D103FD07C95385FFAB0CACBC86 |
| Salt = 'WindowCleaner' | E6F9DCB1D07E5412135120C0257BAA1A27659D41DC7<br>7FE2DE4C345E23CB973415F8DFDFFF6AA7F0AE0BDD<br>61560FB028EFEDF2B5422B40E5EE040A0223D16F06F |
| Salt = 'a0w8hsdfas8ls587uas87' | 5AA762E7C83CFF223B5A00ADA939FBD186C4A2CD01<br>1B0A7FE7AF86B8CA5420C7A47B52AFD2FA6B9BB172<br>22ACF32B3E13F8C436447C36364A5E2BE998416A103A |
| Salt = '8ash87123klnf9d8dq3w' | 8058D43195B1CF2794D012A86AC809BFE73254A82C8C<br>E6C10256D1C46B9F45700D040A6AC6290746058A63E5<br>0AAF8C87ABCD5C3AA00CDBDB31C10BA6D12A1A7 |

Tuz değeri oluşturmanın bir yolu, aşağıdaki **örnek 12.4'te** gösterildiği gibi bir sözde rasgele sayı üreteci kullanmaktır.

**Örnek 12.4**

```
private int minSaltSize = 8;
 private int maxSaltSize = 24;
 private int saltSize;

 private byte[] GetSalt(string input) {
      byte[] data;
      byte[] saltBytes;
      RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
      saltBytes = new byte[saltSize];
      rng.GetNonZeroBytes(saltBytes);
      data = Encoding.UTF8.GetBytes(input);
      byte[] dataWithSaltBytes =
        new byte[data.Length + saltBytes.Length];
      for (int i = 0; i < data.Length; i++)
        dataWithSaltBytes[i] = data[i];
      for (int i = 0; i < saltBytes.Length; i++)
        dataWithSaltBytes[data.Length + i] = saltBytes[i];
      return dataWithSaltBytes;
}
```

Bir tuz değerinin kriptografik olarak güvenli bir rastgelelik niteliğine sahip olması gerekmediğini unutmayın. En iyi uygulamalar, tuzu oluşturmak için kriptografik bir fonksiyon kullanmak, her hash değeri için bir tuz değeri oluşturmak ve minimum 128 bitlik (16 karakter) bir değer kullanmaktır. Bitler maliyetli değildir, bu nedenle performanstan bir şeyler kazanacağınızı düşünerek birkaç bit tasarruf etmeyin, bunun yerine 256 bitlik bir tuz değeri kullanın. Şiddetle tavsiye edilir.

### En İyi Uygulamalar

Sektörün önde gelen Kriptografları MD5 ve SHA-1'in hiçbir uygulama için kullanılmamasını tavsiye etmektedir. Birleşik Devletler FEDERAL BİLGİ İŞLEM STANDARTLARI YAYINI (FIPS) yedi kriptografik hash algoritmasını belirtir - SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 ve SHA-512/256 federal kullanım için onaylanmıştır.

FIPS bilgi teknolojisi endüstrisi tarafından da yaygın olarak benimsendiği için kod gözden geçiricisi bu standardı dikkate almalıdır.

MD5 ve SHA-1 kullanıldığında kodu gözden geçiren kişi kırmızı bayrak göstermeli ve daha uygun diğer hash fonksiyonları yerine neden bu fonksiyonların kullanıldığını anlamak için bir risk değerlendirmesi yapılmalıdır. FIPS, MD5'in yalnızca algoritma tarafından hiçbir güvenliğin sağlanmadığı onaylı bir anahtar taşıma şemasının bir parçası olarak kullanıldığında kullanılmasına izin verir.

Aşağıdaki **örnek 12.5,** bir uygulama için genel bir hash özelliği uygulayabilecek örnek bir işlevi göstermektedir.

**Örnek 12,5**

Uygulama Kodu Dosyası:
<add key="HashMethod" value="SHA512"/>

C# Kodu:

```
1: preferredHash = HashAlgorithm.Create((string)ConfigurationManager.AppSettings["HashMethod"]);
2:
3: hash = computeHash(preferredHash, testString);
4:
5: private string computeHash(HashAlgorithm myHash, string input) {
6:     byte[] data;
7:     data = myHash.ComputeHash(Encoding.UTF8.GetBytes(input));
8:     sb = new StringBuilder();
9:     for (int i = 0; i < data.Length; i++) {
10:        sb.Append(data[i].ToString("x2"));
11:    }
12:    return sb.ToString();
13: }
```

**1. satır,** kullanacağımız hashing algoritmasını yapılandırma dosyasından almamızı sağlar. Eğer makine yapılandırmasını kullanırsak dosya uygulamamız uygulamaya özel yerine sunucu çapında olacaktır.

**Satır 3,** yapılandırma değerini kullanmamıza ve hashing işlevi seçimimize göre ayarlamamıza olanak tanır. ComputHash SHA-256 veya SHA-512 olabilir.

**Referanslar**

**http://valerieaurora.org/hash.html (Kriptografik hash fonksiyonlarının ömürleri)**

**http://docs.oracle.com/javase/6/docs/api/java/security/SecureRandom.html**

**http://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider.aspx**

**http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf**

**Ferguson ve Schneier (2003) Practical Cryptography (bkz. Bölüm 6; kısım 6.2 Real Hash Functions)**

### 12.6 Saldırı yüzeyinin azaltılması

Bir uygulamanın Saldırı Yüzeyi, giriş/çıkış noktalarının, kullanıcıların rollerinin/yetkilerinin ve uygulama içinde tutulan verilerin hassasiyetinin bir açıklamasıdır. Örneğin, oturum açma ekranları, HTML formları, dosya yükleme ekranları gibi giriş noktalarının tümü uygulamaya bir risk düzeyi getirir. Kod yapısının da Saldırı Yüzeyinin bir parçasını oluşturduğunu, kimlik doğrulama veya kripto vb. kontrol eden kodun uygulamadaki kritik işlevler tarafından kullanıldığını unutmayın.

**Açıklama**

Bir yazılım ortamının saldırı yüzeyi, bir saldırganın bir uygulamayı manipüle etmeye çalışabileceği giriş noktalarının bir tanımıdır ve tipik olarak tüm giriş noktalarının (arayüzler) belirtildiği bir sistem diyagramı şeklini alır.

Michael Howard (Microsoft'ta) ve diğer araştırmacılar, bir uygulamanın Saldırı Yüzeyini ölçmek ve zaman içinde Saldırı Yüzeyindeki değişiklikleri izlemek için Göreceli Saldırı Yüzeyi Bölümü (RSQ) adı verilen bir yöntem geliştirdiler.

Uygulama Saldırı Yüzeyinin, muhtemelen daha önceki bazı tehdit modelleme çalışmaları veya Mimari Risk Analizi yoluyla zaten bilindiği varsayılmaktadır. Bu nedenle giriş ve çıkış noktaları bilinmektedir, uygulama içindeki verilerin hassasiyeti anlaşılmıştır ve sistemin çeşitli kullanıcıları ve yetkileri, işlevler ve verilerle ilişkili olarak haritalandırılmıştır.

Kod incelemesi açısından bakıldığında amaç, incelenen değişikliğin Saldırı Yüzeyini gereksiz yere artırmadığından emin olmaktır. Örneğin, kod değişikliği daha önce sadece HTTPS kullanılırken aniden HTTP mi kullanıyor? Kodlayıcı, önceden var olan (ve iyi alıştırılmış/test edilmiş) merkezi kripto fonksiyonları havuzunu kullanmak yerine kendi hash fonksiyonunu yazmaya mı karar veriyor? Bazı geliştirme ortamlarında Saldırı Yüzeyi değişiklikleri, bu tür ayrıntılar yakalanırsa tasarım aşamasında kontrol edilebilir, ancak kod incelemesinde gerçek uygulama koda yansıtılır ve bu tür Saldırı Yüzeyi açıkları tespit edilebilir.

Ayrıca uygulamayı tarayarak Saldırı Yüzeyinin bir resmini de oluşturabilirsiniz. Web uygulamaları için OWASP Zed Attack Proxy Project (ZAP), Arachni, Skipfish, w3af gibi bir araç ya da uygulamanızı taramak ve uygulamanın web üzerinden erişilebilen kısımlarını haritalamak için birçok ticari dinamik test ve güvenlik açığı tarama aracı ya da hizmetinden birini kullanabilirsiniz. Saldırı Yüzeyinin bir haritasını çıkardıktan sonra, yüksek riskli alanları belirleyin ve ardından hangi engelleyici kontrollere sahip olduğunuzu anlayın.

Kod ve veri yedeklerinin (çevrimiçi ve çevrimdışı ortamlarda) bir sistemin Saldırı Yüzeyi'nin önemli ancak genellikle göz ardı edilen bir parçası olduğunu unutmayın. Güvenli yazılım yazarak ve altyapıyı güçlendirerek verilerinizi ve IP'nizi korumanız, yedeklerinizi korumayarak her şeyi kötü adamlara teslim ederseniz boşa gidecektir.

### Ne İncelenmeli

Kod modüllerini Saldırı Yüzeyi bakış açısıyla incelerken dikkat edilmesi gereken bazı yaygın sorunlar şunlardır:

- Kod değişikliği saldırı yüzeyini değiştiriyor mu? Değişikliği uygulamanın mevcut Saldırı Yüzeyine uygulayarak yeni portlar açıyor veya yeni girdiler kabul ediyor mu? Eğer değiştiriyorsa, değişiklik saldırı yüzeyini artırmayacak şekilde yapılabilir mi? Daha iyi bir uygulama mevcutsa, bu önerilmelidir, ancak kodu Saldırı Yüzeyini artırmadan uygulamanın bir yolu yoksa, işletmenin artan riski bildiğinden emin olun.

- Bu özellik HTTPS yerine gereksiz yere HTTP mi kullanıyor?

- İşlev, kimliği doğrulanmamış kullanıcılar tarafından kullanılabilecek mi? İşlevin çağrılması için kimlik doğrulaması gerekmiyorsa, saldırganların arayüzü kullanma riski artar. İşlev, diğer meşru kullanıcıların hizmetlerini reddetmek için kullanılabilecek bir arka uç görevi çağırıyor mu?
  - Örneğin, işlev bir dosyaya yazıyorsa veya bir SMS gönderiyorsa ya da CPU yoğun bir hesaplamaya neden oluyorsa, bir saldırgan işlevi saniyede birçok kez çağırmak ve meşru kullanıcıların bu göreve erişimini engellemek için bir komut dosyası yazabilir mi?

- Aramalar kontrol ediliyor mu? Arama, genellikle veritabanını bazı kriterler için sorguladığı ve sonuçları döndürdüğü için riskli bir işlemdir, saldırgan sorguya SQL enjekte edebilirse, amaçlanandan daha fazla veriye erişebilir.

- Önemli veriler önemsiz verilerden ayrı olarak mı saklanıyor (DB, dosya depolama, vb.)? Değişiklik, kimliği doğrulanmamış kullanıcıların kullanıcı adı/parola tablosuyla aynı bölümdeki bir veritabanı tablosunda herkese açık mağaza konumlarını aramasına izin verecek mi? Veritabanı bilgilerine yönelik riski azaltmak için bu mağaza konumu verileri farklı bir veritabanına veya farklı bir bölüme mi konulmalıdır?

• Dosya yüklemelerine izin veriliyorsa, kimlik doğrulaması yapılıyor mu? Hız sınırlaması var mı? Her yükleme için maksimum dosya boyutu veya her kullanıcı için toplam dosya boyutu var mı? Uygulama dosya yüklemelerini belirli dosya türleriyle kısıtlıyor mu (MIME verilerini veya dosya sonekini kontrol ederek). Uygulama virüs kontrolü yapacak mı?

• Yüksek ayrıcalığa sahip yönetici kullanıcılarınız varsa, eylemleri a) günlüğü silemeyecekleri / değiştiremeyecekleri ve b) eylemlerini inkar edemeyecekleri şekilde günlüğe kaydediliyor / izleniyor mu?

o Olmaması gereken hassas verilere erişip erişmediklerini tespit etmek için herhangi bir alarm veya izleme var mı? Bu sadece yöneticiler için değil, tüm kullanıcı türleri için geçerli olabilir.

• Değişiklikler mevcut karşı önlemler veya güvenlik kodu ile uyumlu olacak mı yoksa yeni kod/karşı önlemler geliştirilmesi gerekecek mi?

• Değişiklik, mevcut bir güvenlik modülünü yeniden kullanmak veya genişletmek yerine merkezi olmayan bir güvenlik kodu modülü getirmeye mi çalışıyor?

• Değişiklik, saldırı yüzeyini karmaşıklaştıracak gereksiz kullanıcı seviyeleri veya yetkileri ekliyor mu?

• Değişiklik PII veya gizli verileri depoluyorsa, tüm yeni bilgiler kesinlikle gerekli mi? Veriler hiç kullanılmayacaksa, milyonlarca kişinin sosyal güvenlik numaralarını depolayarak bir uygulamanın riskini artırmanın çok az değeri vardır.

• Uygulama yapılandırması, saldırı yüzeyinin yapılandırma ayarlarına bağlı olarak büyük ölçüde değişmesine neden oluyor mu ve bu yapılandırmanın kullanımı basit mi ve saldırı yüzeyi genişlediğinde yöneticiyi uyarıyor mu?

• Değişiklik, saldırı yüzeyini azaltacak farklı bir şekilde yapılabilir mi, yani yardım öğelerini aranabilir hale getirmek ve yardım öğesi metnini ana kullanıcı adı / şifre deposunun yanında bir veritabanı tablosunda saklamak yerine, HTML sayfalarında statik yardım metni sağlamak 'yardım' arayüzü aracılığıyla riski azaltır.

• Sunucuda depolanması gereken bilgiler istemcide mi depolanıyor?

### 1.2.3 Referanslar

**https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet**

**https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project**

**http://www.cs.cmu.edu/~wing/publications/Howard-Wing03.pdf**


A7

# A7 EKSIK FONKSIYON SEVIYESI ERIŞIM KONTROLÜ

Çoğu web uygulaması, söz konusu işlevi kullanıcı arayüzünde görünür hale getirmeden önce işlev düzeyi erişim haklarını doğrular. Ancak uygulamaların her bir işleve erişildiğinde sunucuda da aynı erişim kontrollerini gerçekleştirmesi gerekir. İstekler doğrulanmazsa, saldırganlar uygun yetkilendirme olmadan işlevselliğe erişmek için istekleri taklit edebileceklerdir.

## 13.1 Yetkilendirme

Yetkilendirme, kimlik doğrulama kadar önemlidir. Uygulama işlevlerine erişim ve tüm verilere erişim yetkilendirilmelidir. Veri erişim yetkilendirmesi için uygulama mantığı, verinin kimliği doğrulanmış kullanıcıya ait olup olmadığını veya kullanıcının bu veriye erişip erişemeyeceğini kontrol etmelidir.

Güvenlik kontrollerinin yerleştirilmesi, bir uygulama tasarımında hayati bir inceleme alanıdır. Yanlış yerleştirme, uygulanan güvenlik kontrollerini işe yaramaz hale getirebilir, bu nedenle uygulama tasarımının gözden geçirilmesi ve bu kontrollerin doğruluğunun belirlenmesi önemlidir. Birçok web uygulaması tasarımı, gelen tüm istekleri dinleyen ve kontrolü uygun form/iş süreç mantığına devreden merkezi bir denetleyiciye sahip Model-Görünüm-Denetleyici (MVC) kavramına dayanmaktadır. Nihayetinde kullanıcıya bir görünüm sunulur. Böyle katmanlı bir tasarımda, bir talebin işlenmesine dahil olan çok sayıda varlık olduğunda, geliştiriciler güvenlik kontrollerini yanlış yere yerleştirerek hata yapabilirler, örneğin bazı uygulama geliştiricileri yetkilendirme kontrolleri için doğru yerin "görünüm" olduğunu düşünmektedir.

Yetkilendirme sorunları, bir web uygulamasındaki çok çeşitli katmanları kapsar; bir kullanıcının uygulama katmanında uygulamanın belirli bir işlevine erişmek için işlevsel yetkilendirmesinden, kalıcılık katmanındaki Veritabanı erişim yetkilendirmesine ve en az ayrıcalık sorunlarına kadar.

Uygulamaların çoğunda istek parametreleri ya da URL'ler işleme mantığını belirleyen yegane unsurlar olarak hizmet vermektedir. Böyle bir senaryoda, bu tür tanımlamalar için kullanılan istek unsurları, uygulamadaki kısıtlı kaynaklara veya sayfalara erişim elde etmek için manipülasyon saldırılarına maruz kalabilir.

Yetkilendirmeyi uygulamak için iki ana tasarım yöntemi vardır: Rol Tabanlı Erişim Kontrolü (RBAC) ve Erişim Kontrol Listeleri (ACL'ler). RBAC, kullanıcıları rollere ve ardından rolleri izinlere atarken kullanılır. Bu, gerçek sistem yetkilendirmesinin daha mantıksal bir modellemesidir. Ayrıca, yöneticilerin rol-izinlerini ince eleyip sık dokumalarına ve yeniden kontrol etmelerine olanak tanırken, her rolün sahip olması gereken izinlere sahip olduğundan (daha fazla ya da daha az değil) emin olmalarını sağlar.

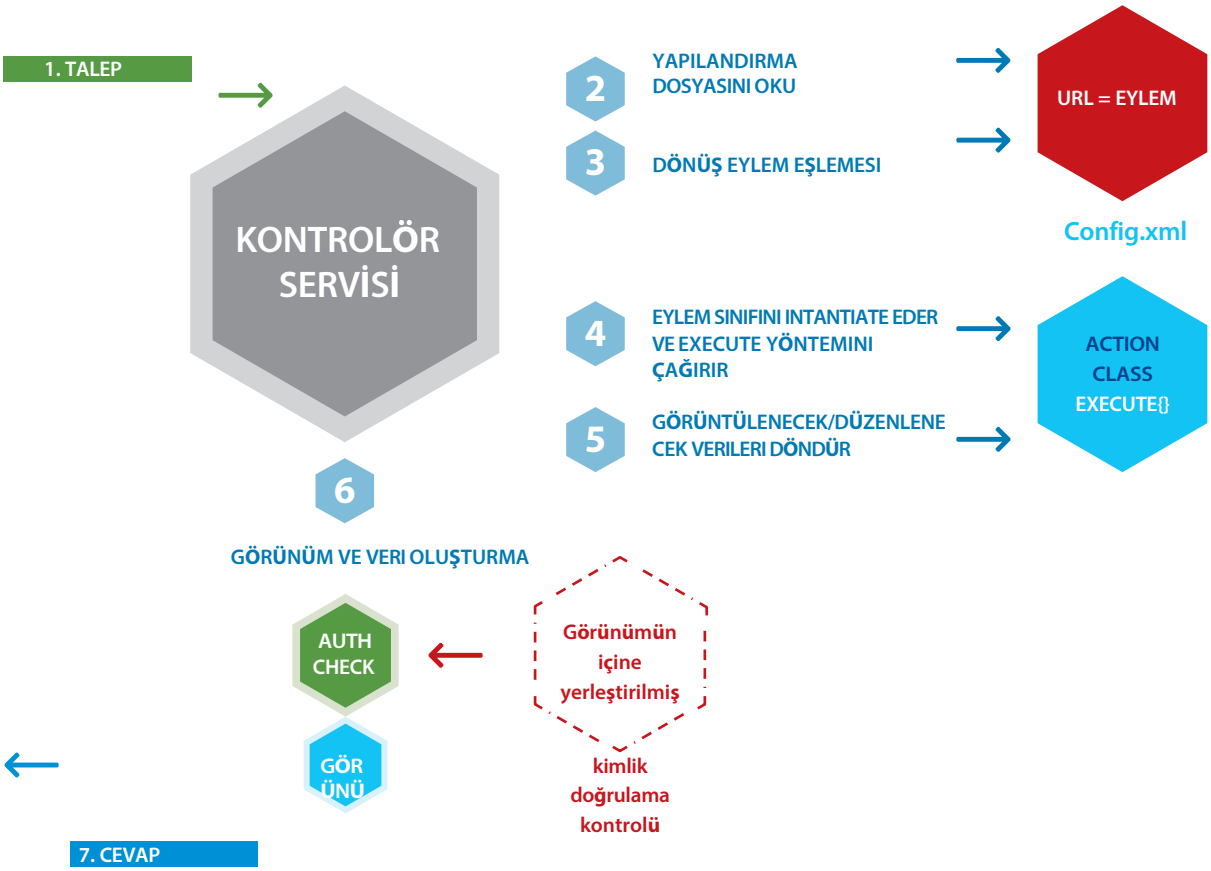
Bu nedenle kullanıcıların rollere atanması insan hatası olasılığını azaltmalıdır. Birçok web çerçevesi, oturum açan kullanıcılara roller atanmasına izin verir ve özel kod, mevcut kullanıcı rolüne dayalı olarak işlevselliği yetkilendirmek için oturum bilgilerini kontrol edebilir.

## 13.2 Açıklama

Kullanıcıları sayfa/görünüm düzeyinde kısıtlamak mantıklı görünmektedir, bu kullanıcılar uygulamada herhangi bir işlem gerçekleştiremeyeceklerdir. Peki ya yetkisiz bir kullanıcı bir sayfa/görünüm talep etmek yerine uygulamada herhangi bir veri eklemek/değiştirmek gibi dahili bir eylem talep etmeye çalışırsa? Bu işlem gerçekleştirilecek ancak sonuçta ortaya çıkan görünüm kullanıcıya reddedilecektir; çünkü kusur, uygulamalarda yalnızca görünüm tabanlı bir erişim kontrolüne sahip olunmasında yatmaktadır. İş mantığı işlemlerinin çoğu (bir istek için) "görünüm" yürütülmeden önce yapılır. Dolayısıyla, herhangi bir eylemi işleme isteği yetkilendirme olmadan başarılı bir şekilde gerçekleştirilebilir.

MVC tabanlı bir sistemde bu konu aşağıdaki **şekil 11'**de gösterilmiştir; burada kimlik doğrulama kontrolü görünüm eyleminde önceden girilmiştir.

**Şekil 11:** MVC Erişim Kontrolü




Bu örnekte ne kontrolör servlet'i (merkezi işlem birimi) ne de eylem sınıfları herhangi bir erişim kontrolüne sahip değildir. Kullanıcı, kimlik doğrulaması olmadan kullanıcı bilgilerini ekleme gibi dahili bir eylem talep ederse, bu eylem işlenecektir, ancak tek fark, sonuçta ortaya çıkan görünüm kullanıcıya izin verilmeyeceği için kullanıcıya bir hata sayfası gösterilmesidir. Benzer bir kusur, geliştiricilerin POSTBACK'leri ve kimlik doğrulama kontrollerini işlemek için kodu karıştırma eğiliminde olduğu ASP.NET uygulamalarında da görülmektedir. Genellikle ASP.NET sayfalarındaki kimlik doğrulama kontrolünün aşağıda belirtildiği gibi POSTBACK'ler için uygulanmadığı görülmektedir. Burada, bir saldırgan kimlik doğrulaması olmadan sayfaya erişmeye çalışırsa bir hata sayfası oluşturulacaktır. Bunun yerine, saldırgan kimlik doğrulaması olmadan doğrudan dahili bir POSTBACK isteği göndermeye çalışırsa başarılı olacaktır.

Yapılandırma dosyalarında kullanılmayan veya beyan edilmemiş eylemler veya işlevler bulunabilir. Uygulamada geçerli özellikler olarak ifşa edilmeyen (veya test edilmeyen) bu tür yapılandırmalar saldırı yüzeyini genişletir ve uygulama için riski artırır. Bir yapılandırma dosyasında bulunan kullanılmayan bir yapılandırma **örnek 13.1**'de gösterilmiştir; burada dosyanın sonundaki 'TestAction' test döngüsünden kalmıştır ve harici kullanıcılara açık olacaktır. Bu eylemin her sürümde kontrol edilmemesi ve bir güvenlik açığını ortaya çıkarması muhtemeldir.

**Örnek 13.1**

```
<haritalama>
 <url>/InsecureDesign/action/AddUserDetails</url>
 <action>Action.UserAction</action>
```

```
 <success>JSP_WithDesign/Success.jsp</success>
</mapping>

<haritalama>
 <url>/InsecureDesign/action/ChangePassword</url>
 <action>Action.ChangePasswordAction</action>
 <success>JSP_WithDesign/Success.jsp</success>
</mapping>

<haritalama>
 <url>/InsecureDesign/action/test</url>
 <action>Action.TestAction</action>
 <success>JSP_WithDesign/Success.jsp</success>
</mapping>
```

Günümüzde tasarım çerçevelerinin çoğunda görülen bir diğer popüler özellik de, istek parametrelerinin doğrudan ilgili iş/komut nesnesinin değişkenlerine bağlandığı veri bağlamadır. Buradaki bağlama, bu tür sınıfların örnek değişkenlerinin, adlarına bağlı olarak istek parametresi değerleriyle otomatik olarak başlatılacağı anlamına gelir. Bu tasarımla ilgili sorun, iş nesnelerinin istek parametrelerine bağlı olmayan değişkenlere sahip olabilmesidir. Bu değişkenler fiyat, maksimum limit, rol vb. gibi statik değerlere sahip veya bazı sunucu tarafı işleme mantığına bağlı anahtar değişkenler olabilir. Bu tür senaryolardaki bir tehdit, bir saldırganın istekte ek parametreler sağlayabilmesi ve iş nesnesi sınıfının açıkta olmayan değişkenleri için değerler bağlamaya çalışmasıdır. Bu durumda saldırgan, istekte ek bir "fiyat" parametresi gönderebilir ve bu parametre iş nesnesindeki açıkta olmayan "fiyat" değişkenine bağlanarak iş mantığını manipüle edebilir.

### Ne İncelenmeli

Herhangi bir iş mantığını işlemeden önce ve ASP.NET uygulamaları söz konusu olduğunda POSTBACK'lerden bağımsız olarak tüm doğrulama kontrollerinin yerleştirilmesi zorunludur. Kimlik doğrulama kontrolü gibi güvenlik kontrolleri herhangi bir istek işlenmeden önce yerleştirilmelidir.

.NET MVC 3, global.asax'ta uygulamadaki URL'lere erişimi varsayılan olarak reddetmek için kullanılabilecek RegisterGlobalFilters adlı bir yöntem sunduğundan, yetkilendirme MVC 3 ve üzerinde uygulandığında filtrelerin kullanılması önerilir.

### Örnek 13.2

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
  filters.Add(new HandleErrorAttribute());
  filters.Add(new System.Web.Mvc.AuthorizeAttribute());
}
```

MVC3/4 .NET'i incelerken yetkilendirmenin nasıl uygulandığına bir göz atmanız önerilir. Yukarıdaki satır, "filters.Add(new System.Web.Mvc.AuthorizeAttribute());" varsayılan olarak geçerli bir yetkilendirme olmadan herhangi bir isteğe erişimi reddeder.

Oturum. Bu uygulandığında, kayıt sayfası, genel karşılama sayfası veya oturum açma sayfası gibi belirli sayfalara yetkisiz erişim sağlamamız gerekebilir.

"AllowAnonymous" yönergesi, geçerli oturum gerekmeden genel sayfalara erişim sağlamak için kullanılır. Kod aşağıdaki gibi görünebilir:

**Örnek 13.3**

```
[AllowAnonymous]
  public ActionResult LogMeIn(string returnUrl)
```

**Yetkilendirme için kodu gözden geçirirken aşağıdaki hususlar kontrol edilebilir:**

• Her giriş noktası yetkilendirilmelidir. Her fonksiyon yetkilendirilmelidir.

• Yetkilendirme kontrolleri verimli olmalı ve tutarlı bir şekilde uygulanabilmesi için merkezi bir kod tabanında uygulanmalıdır.

• Yetkilendirmenin başarısız olduğu durumlarda, bir HTTP 403 yetkili değil sayfası döndürülmelidir.

• RBAC kullanırken, uygulamanın sistemin o anda sağlanmış kullanıcıları ve ilişkili rolleri hakkında rapor vermesinin bir yolu olmalıdır. Bu, işletmenin sisteme kullanıcı erişimini periyodik olarak denetlemesine ve doğru olduğundan emin olmasına olanak tanır. Örneğin, bir kullanıcı sistemde yönetici olarak sağlanmışsa, daha sonra bu kullanıcı başka bir departmanda iş değiştirirse, yönetici rolünün artık uygun olmaması söz konusu olabilir.

• Bir kullanıcının rolünü değiştirmek veya kaldırmak için kolay bir yöntem olmalıdır (RBAC sistemlerinde). Bir kullanıcının bir role eklenmesi, değiştirilmesi veya rolden çıkarılması denetim günlükleriyle sonuçlanmalıdır.

• Daha yüksek riskli roller için, bu rollerin eklenmesi, değiştirilmesi ve silinmesi birden fazla yetkilendirme seviyesini içermelidir (örneğin, yapımcı/denetleyici), bu uygulamanın kendi içinde veya bazı merkezi rol uygulamaları aracılığıyla izlenebilir. Rolleri kontrol eden sistemin hem işlevselliği hem de kodu inceleme kapsamının bir parçası olmalıdır.

• Tasarım düzeyinde rol yelpazesini basit tutmaya çalışın. Çoklu izin seviyeleri/rolleri olan uygulamalar genellikle beklenmedik ayrıcalıklarla sonuçlanan çakışan izin setleri olasılığını artırır.

• Kalın istemcili uygulama mimarilerinde (yani mobil uygulamalar veya bilgisayarda çalışan ikili dosyalar), bir saldırgan tarafından atlanabileceğinden, istemci kodunda herhangi bir yetkilendirme gerçekleştirmeye çalışmayın. Tarayıcı tabanlı uygulamalarda JavaScript'te herhangi bir yetkilendirme kararı vermeyin.

• Yetkilendirme kararlarını asla güvenilmeyen verilere dayandırmayın. Örneğin, bir kullanıcının sahip olacağı yetkilendirme seviyesini belirlemek için istemci isteğinden bir başlık veya gizli alan kullanmayın, bu yine bir saldırgan tarafından manipüle edilebilir.

• Yetkilendirmenin bir işlevin her aşamasında kontrol edildiği 'tam aracılık' ilkesini izleyin. Örneğin, bir uygulamada bir ürün satın almak için göz atılacak dört sayfa varsa (browse.html, basket.html, inputPay- ment.html, makePayment.html), yalnızca ilk sayfada bir kontrol gerçekleştirmek yerine her sayfada ve sayfalar içindeki aşamalarda kullanıcı yetkisini kontrol edin.

• Varsayılan olarak herhangi bir sayfaya erişimi reddedin ve ardından roller/ACL kurallarına dayalı olarak erişime açıkça izin vermek için yetkilendirme mantığını kullanın.

• Yetkilendirme verileri nerede tutulmalı, DB mi yoksa oturum mu?

• Tüm gereksiz/test/açıkta kalan iş mantığı yapılandırmalarını dosyadan kaldırın

• İş/form/komut nesneleri yalnızca kullanıcı girdilerine bağlı olan örnek değişkenlere sahip olmalıdır.


A8

# A8 SITELER ARASI ISTEK SAHTECILIĞI (CSRF)

CSRF saldırısı, oturum açmış bir kurbanın tarayıcısını, kurbanın oturum çerezini ve otomatik olarak dahil edilen diğer kimlik doğrulama bilgilerini içeren sahte bir HTTP isteğini savunmasız bir web uygulamasına göndermeye zorlar. Bu, saldırganın kurbanın tarayıcısını, savunmasız uygulamanın kurbandan gelen meşru talepler olduğunu düşündüğü talepler oluşturmaya zorlamasına olanak tanır.

## 14.1 Açıklama

CSRF, son kullanıcıyı kimliğinin doğrulandığı bir web uygulamasında istenmeyen eylemler gerçekleştirmeye zorlayan bir saldırıdır. Saldırgan, biraz sosyal mühendislik yardımıyla (e-posta/sohbet yoluyla bir bağlantı göndermek gibi) bir web uygulamasının kullanıcılarını saldırganın seçtiği eylemleri gerçekleştirmeye zorlayabilir. Başarılı bir CSRF uygulaması, normal bir ayrıcalıklı kullanıcı durumunda son kullanıcı verilerini ve korunan işlevselliği tehlikeye atabilir. Hedeflenen son kullanıcı yönetici hesabıysa, bu tüm web uygulamasını tehlikeye atabilir.

Başarılı bir siteler arası istek sahteciliği saldırısının etkisi savunmasız uygulama tarafından açığa çıkarılan yeteneklerle sınırlıdır. Örneğin, bu saldırı para transferi, parola değiştirme ya da kullanıcı bağlamında bir öğe satın alma ile sonuçlanabilir. Aslında CSRF saldırıları bir saldırgan tarafından hedef kullanıcının bilgisi olmadan, en azından yetkisiz işlev gerçekleştirilinceye kadar, hedef sistemin tarayıcısı üzerinden bir işlevi (para transferi, form gönderme vb.) yerine getirmesini sağlamak için kullanılır.

CSRF, kötü amaçlı içeriğin güvenilir bir web sitesi tarafından şüphelenmeyen bir kurbana sunulmasına neden olan XSS (Cross Site Scripting) ile aynı değildir. Siteler Arası İstek Sahteciliği (CSRF, diğer adıyla C-SURF veya Confused-Deputy), saldırgan hedefin web tabanlı bir sistemde kimliğinin doğrulandığını biliyorsa yararlı kabul edilir. Yalnızca hedef sisteme giriş yapmışsa çalışırlar ve bu nedenle küçük bir saldırı ayak izine sahiptirler. Kullanıcı tarafından işlem yetkisi gerekmemesi gibi diğer mantıksal zayıflıkların da mevcut olması gerekir. Aslında CSRF saldırıları, bir saldırgan tarafından, hedef kullanıcının bilgisi olmadan, en azından yetkisiz işlev gerçekleştirilinceye kadar, hedefin tarayıcısı aracılığıyla hedef sistemin bir işlevi (Fon Transferi, Form gönderimi vb.) yerine getirmesini sağlamak için kullanılır. Birincil hedef, web uygulamalarındaki "kullanım kolaylığı" özelliklerinin istismar edilmesidir (Tek tıkla satın alma).

Başarılı CSRF istismarlarının etkileri, kurbanın rolüne bağlı olarak büyük ölçüde değişir. Normal bir kullanıcı hedeflendiğinde, başarılı bir CSRF saldırısı son kullanıcı verilerini ve bunlarla ilişkili işlevleri tehlikeye atabilir. Hedeflenen son kullanıcı bir yönetici hesabıysa, bir CSRF saldırısı tüm Web uygulamasını tehlikeye atabilir. Saldırıya uğrama olasılığı daha yüksek olan siteler, topluluk Web siteleri (sosyal ağ, e-posta) veya bunlarla ilişkili yüksek dolar değerli hesaplara sahip sitelerdir (bankalar, borsa aracı kurumları, fatura ödeme hizmetleri). Bu saldırı, kullanıcı güçlü şifreleme (HTTPS) kullanan bir Web sitesine giriş yapmış olsa bile gerçekleşebilir. Sosyal mühendisliği kullanan bir saldırgan, belirli bir 'görev URL'si' talep etmek için bir e-postaya veya Web sitesine kötü amaçlı HTML veya JavaScript kodu yerleştirir. Görev daha sonra kullanıcının bilgisi dahilinde ya da bilgisi dışında, doğrudan ya da bir Siteler Arası Komut Dosyası açığı (örn: Samy MySpace Worm) kullanılarak yürütülür.

### Nasıl Çalışırlar

CSRF saldırıları, kimliği doğrulanmış bir kullanıcının tarayıcısından uygulamaya sahte bir HTTP isteği göndererek çalışır ve ardından hedef kullanıcı tarafından verilen yetki olmadan bir işlem gerçekleştirir. Kullanıcının kimliği doğrulandığı ve kullanıcının tarayıcısı tarafından hedef uygulamaya anlamlı bir HTTP isteği gönderildiği sürece, uygulama isteğin kaynağının geçerli bir işlem mi yoksa kullanıcı uygulamaya kimlik doğrulaması yaparken kullanıcı tarafından tıklanan bir bağlantı mı (örneğin bir e-postada) olduğunu bilmez. Kullanıcı tarayıcısından gelen istek otomatik olarak kimlik doğrulamanın temeli olan 'Cookie' başlığını içereceğinden isteğin kimliği doğrulanacaktır. Yani

Bir saldırgan, kurbanın bir ürün satın almak gibi niyet etmediği eylemleri gerçekleştirmesini sağlar. **Örnek 14.1'de, bir** dizi bilet satın almak için bir bilet satıcısına gönderilen bir HTTP POST örneği gösterilmektedir.

**Örnek 14.1**

```
POST http://TicketMeister.com/Buy_ticket.htm HTTP/1.1
Ana bilgisayar: ticketmeister
Kullanıcı-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O;) Firefox/1.4.1
Çerez: JSPSESSIONID=34JHURHD894LOP04957HR49I3JE383940123K
ticketId=ATHX1138&to=PO BOX 1198 DUBLIN 2&amount=10&date=11042008
Satıcının yanıtı biletlerin satın alındığını onaylamak içindir:

HTTP/1.0 200 TAMAM
Tarih Fri, 02 May 2008 10:01:20 GMT
Sunucu: IBM_HTTP_Server
Content-Type: text/xml;charset=ISO-8859-1
Content-Language: en-US
X-Cache: MISS from app-proxy-2.proxy.ie
Bağlantı: close

<?xml version="1.0" encoding="ISO-8859-1"?>
<pge_data> Bilet Satın Alındı, Siparişiniz için teşekkür ederiz.
</pge_data>
```

### Ne İncelenmeli

Bu sorunu tespit etmek kolaydır, ancak uygulamanın işlevselliği etrafında kullanıcıyı bir CSRF girişimine karşı uyarabilecek telafi edici kontroller olabilir. Uygulama iyi biçimlendirilmiş bir HTTP isteğini kabul ettiği ve istek uygulamanın bazı iş mantığına uyduğu sürece CSRF çalışacaktır.

Sayfa oluşturmayı kontrol ederek, kullanıcının tarayıcısında uygulama tarafından oluşturulan bağlantılara herhangi bir benzersiz tanımlayıcı eklenip eklenmediğini görmemiz gerekir. Bir HTTP isteğini kullanıcıya bağlamak için her HTTP isteğiyle ilgili benzersiz bir tanımlayıcı yoksa, savunmasız durumdayız demektir. Oturum kimliği yeterli değildir, çünkü bir kullanıcı sahte bir bağlantıya tıklarsa, kullanıcının kimliği zaten doğrulandığından oturum kimliği otomatik olarak gönderilecektir.

### İşe Yaramayan Önleme Tedbirleri

Saldırganların atlayabileceği CSRF önleme teknikleri örnekleri **tablo 21'**de listelenmiştir, bu önlemler hassas uygulamalarda kullanılmamalı ve kod incelemesinde başarısız olmalıdır.

**Tablo 21:** Csrf Saldırıları İçin Başarısız Karşı Önlemler

| Ölçü | Açıklama |
|---|---|
| Gizli Çerez Kullanma | Gizli olanlar da dahil olmak üzere tüm çerezlerin her istekle birlikte gönderileceğini unutmayın. Tüm kimlik doğrulama belirteçleri, son kullanıcının isteği göndermesi için kandırılıp kandırılmadığına bakılmaksızın gönderilecektir. Dahası, oturum tanımlayıcıları sadece uygulama konteyneri tarafından isteği belirli bir oturum nesnesiyle ilişkilendirmek için kullanılır. Oturum tanımlayıcısı, son kullanıcının isteği gönderme niyetinde olduğunu doğrulamaz. |

| | |
|---|---|
| Yalnızca POST İsteklerini Kabul Etme | Uygulamalar, iş mantığının yürütülmesi için yalnızca POST isteklerini kabul edecek şekilde geliştirilebilir. Yanlış kanı, saldırgan kötü niyetli bir bağlantı oluşturamayacağı için CSRF saldırısının gerçekleştirilemeyeceği yönündedir. Ne yazık ki bu mantık yanlıştır. Bir saldırganın kurbanı sahte bir POST isteği göndermesi için kandırabileceği çok sayıda yöntem vardır, örneğin saldırganın web sitesinde gizli değerlerle barındırılan basit bir form gibi. Bu form JavaScript tarafından otomatik olarak tetiklenebilir veya formun başka bir şey yapacağını düşünen kurban tarafından tetiklenebilir. |
| Salt = 'a0w8hsdfas8ls587uas87' | Çok Adımlı işlemler CSRF'ye karşı yeterli bir önlem değildir. Bir saldırgan tamamlanan işlemin her adımını tahmin edebildiği veya çıkarabildiği sürece CSRF mümkündür. |
| URL Yeniden Yazma | Saldırgan kurbanın oturum kimliğini tahmin edemeyeceği için bu yararlı bir CSRF önleme tekniği olarak görülebilir. Ancak, kullanıcının kimlik bilgileri URL üzerinden açığa çıkar. |

Şifre, para transferi, satın alma, uygulama sadece isteği yerine getirmeyebilir, ancak kullanıcının şifresi veya bant dışı (yani iki faktörlü) bir kimlik doğrulama öğesi için başka bir istekle talebe yeniden yanıt vermelidir.

## CSRF'yi Önleme

İsteğin geçerli bir oturum çerezine sahip olup olmadığını kontrol etmek yeterli değildir, uygulamanın, uygulamaya gönderilen her HTTP isteği ile ilişkilendirilmiş benzersiz bir tanımlayıcıya sahip olması gerekir. CSRF istekleri (bir saldırganın e-postasından gelen) bu geçerli benzersiz tanımlayıcıya sahip olmayacaktır. CSRF isteklerinin bu benzersiz istek tanımlayıcısına sahip olmamasının nedeni, benzersiz kimliğin gizli bir alan olarak veya URL içinde işlenmesi ve bir bağlantı / düğmeye basıldığında HTTP isteğine eklenmesidir. Saldırganın bu benzersiz kimlik hakkında hiçbir bilgisi olmayacaktır, çünkü bu kimlik rastgele ve her bağlantıda, her sayfada dinamik olarak oluşturulur.

**CSRF'yi önlemek için uygulama mantığı şunları içerir:**

**1.** Sayfayı kullanıcıya sunmadan önce benzersiz kimliklerin bir listesi derlenir. Liste, belirli bir sayfadaki tüm bağlantılar için oluşturulan tüm geçerli benzersiz kimlikleri içerir. Benzersiz kimlik, J2EE için SecureRandom gibi güvenli bir rastgele oluşturucudan türetilebilir ve oturumda veya başka bir merkezi önbellekte saklanabilir.

**2.** Kullanıcıya gösterilmeden önce istenen sayfadaki her bağlantıya/forma benzersiz bir ID eklenir.

**3.** Uygulama, HTTP isteğiyle birlikte iletilen benzersiz kimliğin belirli bir istek için geçerli olup olmadığını kontrol eder. HTTP isteğiyle birlikte iletilen benzersiz kimlik belirli bir istek için geçerliyse.

**4.** Benzersiz kimlik mevcut değilse, kullanıcı oturumunu sonlandırır ve kullanıcıya bir hata görüntüler.

**Genel Tavsiye: Eşitleyici Belirteç Modeli**

"Şeffaf ama görünür" bir CSRF çözümünü kolaylaştırmak için, geliştiricilerin Syn- chronizer Token Modelini benimsemeleri teşvik edilmektedir http://www.corej2eepatterns.com/Design/PresoDesign.htm. Eşzamanlayıcı belirteç modeli, kullanıcının mevcut oturumuyla ilişkilendirilen rastgele "meydan okuma" belirteçlerinin oluşturulmasını gerektirir. Bu meydan okuma belirteçleri daha sonra HTML formlarına ve hassas sunucu tarafı işlemleriyle ilişkili bağlantılara eklenir. Kullanıcı bu hassas işlemleri çağırmak istediğinde, HTTP isteği bu meydan okuma belirtecini içermelidir. Daha sonra bu belirtecin varlığını ve doğruluğunu doğrulamak sunucu uygulamasının sorumluluğundadır. Geliştirici, her isteğe bir meydan okuma belirteci ekleyerek, kullanıcının gerçekten istenen istekleri göndermeyi amaçladığını doğrulamak için güçlü bir kontrole sahip olur. Hassas iş işlevleriyle ilişkili HTTP isteklerine gerekli bir güvenlik belirtecinin dahil edilmesi CSRF saldırılarının azaltılmasına yardımcı olur çünkü başarılı bir istismar saldırganın hedef kurbanın oturumu için rastgele oluşturulan belirteci bildiğini varsayar. Bu, saldırganın hedef kurbanın oturum tanımlayıcısını tahmin edebilmesine benzer. Aşağıdaki özet, meydan okuma belirteçlerini istek içine dahil etmek için genel bir yaklaşımı açıklamaktadır.

Bir Web uygulaması bir istek formüle ettiğinde (gönderildiğinde bir isteğe neden olan bir bağlantı veya form oluşturarak veya

kullanıcı tarafından tıklandığında), uygulama "CSRFTo- ken" gibi yaygın bir ada sahip gizli bir girdi parametresi içermelidir. Bu belirtecin değeri, bir saldırgan tarafından tahmin edilemeyecek şekilde rastgele oluşturulmalıdır. Yeterince uzun rastgele bir belirteç oluşturmak için Java uygulamaları için java.security.SecureRandom sınıfından yararlanmayı düşünün. Alternatif oluşturma algoritmaları arasında 256 bit BASE64 kodlu karmaların kullanımı yer alır. Bu oluşturma algoritmasını seçen geliştiriciler, rastgele belirteci oluşturmak için karma haline getirilen verilerde rastgelelik ve benzersizliğin kullanıldığından emin olmalıdır.

**Örnek 14.2**

```
<form action="/transfer.do" method="post">
<input type="hidden" name="CSRFToken"
value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWE...
wYzU1YWQwMTVhM2JmNGYxYjJiMGI4MjJZDE1ZDZ...
MGYwMGEwOA==">
...
</form>
```

Ürünün risk düzeyine bağlı olarak, mevcut oturum için bu belirteci yalnızca bir kez oluşturabilir. Bu belirtecin ilk oluşturulmasından sonra, değer oturumda saklanır ve oturum sona erene kadar sonraki her istek için kullanılır. Son kullanıcı tarafından bir istek gönderildiğinde, sunucu tarafı bileşeni, oturumda bulunan belirteçle karşılaştırıldığında istekteki belirtecin varlığını ve geçerliliğini doğrulamalıdır. Belirteç istekte bulunmazsa veya sağlanan değer oturumdaki değerle eşleşmezse, istek iptal edilmeli, belirteç sıfırlanmalı ve olay devam eden olası bir CSRF saldırısı olarak kaydedilmelidir.

Önerilen bu tasarımın güvenliğini daha da artırmak için CSRF token parametre adını ve / veya değerini her istek için rastgele hale getirmeyi düşünün. Bu yaklaşımın uygulanması, oturum başına belirteçler yerine istek başına belirteçlerin üretilmesiyle sonuçlanır. Ancak bunun kullanılabilirlik sorunlarına yol açabileceğini unutmayın. Örneğin, bir önceki sayfa artık geçerli olmayan bir belirteç içerebileceğinden, "Geri" düğmesi tarayıcı özelliği genellikle engellenir. Bu önceki sayfa ile etkileşim, sunucuda CSRF yanlış pozitif güvenlik olayı ile sonuçlanacaktır. Hangi yaklaşım benimsenirse benimsensin, geliştiricilerin CSRF belirtecini, SSLv3/TLS kullanımı gibi kimliği doğrulanmış oturum tanımlayıcılarını korudukları şekilde korumaları teşvik edilmektedir.

**URL'de Belirtecin Açıklanması**

Bu kontrolün birçok uygulaması, GET (URL) isteklerinin yanı sıra POST isteklerinde de meydan okuma belirtecini içerir. Bu genellikle hassas sunucu tarafı işlemlerinin sayfadaki gömülü bağlantılar veya diğer genel tasarım modellerinin bir sonucu olarak çağrılmasının bir sonucu olarak uygulanır. Bu modeller genellikle CSRF hakkında bilgi sahibi olmadan ve CSRF önleme tasarım stratejilerini anlamadan uygulanmaktadır. Bu kontrol CSRF saldırıları riskini azaltmaya yardımcı olsa da, oturum başına benzersiz belirteç GET istekleri için açığa çıkmaktadır. GET isteklerindeki CSRF belirteçleri potansiyel olarak çeşitli konumlarda sızdırılır: tarayıcı geçmişi, HTTP günlük dosyaları, HTTP isteğinin ilk satırını günlüğe kaydetmek için bir nokta oluşturan ağ cihazları ve korunan site harici bir siteye bağlanırsa Referer başlıkları.

İkinci durumda (Referer başlığının bağlantılı bir site tarafından ayrıştırılması nedeniyle CSRF belirtecinin sızdırılması), bağlantılı sitenin korunan siteye bir CSRF saldırısı başlatması çok kolaydır ve Referer başlığı onlara CSRF belirtecinin yanı sıra siteyi de söylediği için bu saldırıyı çok etkili bir şekilde hedefleyebileceklerdir. Saldırı tamamen javascript'ten çalıştırılabilir, böylece bir sitenin HTML'sine basit bir komut dosyası etiketi eklenmesi bir saldırı başlatabilir (ister orijinal olarak kötü amaçlı bir sitede ister saldırıya uğramış bir sitede olsun). Bu saldırı senaryosunun önlenmesi kolaydır, eğer isteğin kaynağı

HTTPS'dir. Bu nedenle bu saldırı yalnızca HTTPS olan web uygulamalarını etkilemez.
İdeal çözüm, CSRF belirtecini yalnızca POST isteklerine dahil etmek ve durum değiştirme etkisi olan sunucu tarafı eylemlerini yalnızca POST isteklerine yanıt verecek şekilde değiştirmektir. Bu aslında RFC 2616'nın GET istekleri için gerektirdiği şeydir. Sen- sitif sunucu tarafı eylemlerinin yalnızca POST isteklerine yanıt vereceği garanti edilirse, belirteci GET isteklerine dahil etmeye gerek yoktur.

**Viewstate (ASP.NET)**
ASP.NET, ViewState'inizi korumak için bir seçeneğe sahiptir. ViewState, bir sayfanın sunucuya gönderildiğindeki durumunu gösterir. Durum, her sayfaya <form runat="server"> kontrolüyle yerleştirilen gizli bir alan aracılığıyla tanımlanır. Viewstate, bir saldırganın geçerli bir Viewstate'i taklit etmesi zor olduğu için CSRF savunması olarak kullanılabilir. Parametre değerlerinin saldırgan tarafından elde edilmesi veya tahmin edilmesi mümkün olduğundan geçerli bir Viewstate'i taklit etmek imkansız değildir. Bununla birlikte, geçerli oturum kimliği ViewState'e eklenirse, her Viewstate benzersiz hale gelir ve böylece CSRF'ye karşı bağışıklık kazanır.

Sahte posta geri dönüşlerine karşı koruma sağlamak amacıyla Viewstate içindeki ViewStateUserKey özelliğini kullanmak için sayfa türevi sınıfın OnInit sanal yöntemine aşağıdakileri ekleyin (Bu özellik Page.Init olayında ayarlanmalıdır)

**Örnek 14.3**

```
protected override OnInit(EventArgs e) {
   base.OnInit(e);
   if (User.Identity.IsAuthenticated)
   ViewStateUserKey = Session.SessionID; }
```

Seçtiğiniz benzersiz bir değeri kullanarak Viewstate'i bir kişiye anahtarlamak için "(Page.ViewStateUserKey)" kullanın. Bu, Page_Init içinde uygulanmalıdır çünkü Viewstate yüklenmeden önce anahtarın ASP.NET'e sağlanması gerekir. Bu seçenek ASP.NET 1.1'den beri mevcuttur. Ancak, bu mekanizma üzerinde sınırlamalar vardır. Örneğin, ViewState MAC'leri yalnızca POSTback'te kontrol edilir, bu nedenle postback kullanmayan diğer tüm uygulama istekleri CSRF'ye memnuniyetle izin verir.

**Çift Gönderimli Çerezler**
Çerezlerin çift gönderilmesi, hem çerez hem de istek parametresi olarak rastgele bir değer gönderilmesi ve sunucunun çerez değeri ile istek değerinin eşit olup olmadığını doğrulaması olarak tanımlanır.

Bir kullanıcı bir siteye kimlik doğrulaması yaptığında, site (kriptografik olarak güçlü) bir sözde rastgele değer üretmeli ve bunu kullanıcının makinesinde oturum kimliğinden ayrı bir çerez olarak ayarlamalıdır. Sitenin bu değeri herhangi bir şekilde kaydetmesi gerekmez. Site daha sonra her hassas gönderimin bu rastgele değeri gizli bir form değeri (veya başka bir istek parametresi) ve ayrıca bir çerez değeri olarak içermesini zorunlu kılmalıdır. Saldırgan, aynı kaynak politikası uyarınca sunucudan gönderilen hiçbir veriyi okuyamaz veya çerez değerlerini değiştiremez. Bu, bir saldırganın kötü niyetli bir CSRF isteğiyle istediği herhangi bir değeri gönderebileceği, ancak saldırganın çerezde saklanan değeri değiştiremeyeceği veya okuyamayacağı anlamına gelir. Çerez değeri ile istek parametresi veya form değerinin aynı olması gerektiğinden, saldırgan rastgele CSRF değerini tahmin edemediği sürece bir formu başarıyla gönderemeyecektir.

Direct Web Remoting (DWR) Java kütüphanesi sürüm 2.0, çift çerez gönderimini şeffaf bir şekilde uyguladığı için yerleşik CSRF korumasına sahiptir.
Yukarıdaki CSRF önleme yöntemleri, isteklerin kimliğini doğrulamak için gizli bir belirteç tutarak CSRF'yi önlemek için

benzersiz bir belirteç ve Aynı Kaynak İlkesi kullanımına dayanır. Aşağıdaki yöntemler benzer kurallara dayanarak CSRF'yi önleyebilir

CSRF istismarlarının asla kıramayacağı.

**Referer Başlığını Kontrol Etme**
Kendi tarayıcınızda referer başlığını taklit etmek önemsiz olsa da, bir CSRF saldırısında bunu yapmak imkansızdır. Yönlendiriciyi kontrol etmek, gömülü ağ cihazlarında CSRF'yi önlemek için yaygın olarak kullanılan bir yöntemdir çünkü kullanıcı başına bir durum gerektirmez. Bu da yönlendiriciyi, bellek yetersiz olduğunda CSRF önleme için kullanışlı bir yöntem haline getirir. Bu CSRF azaltma yöntemi, bir senkronizasyon belirtecini takip etmek için gerekli olan bir oturum durumu oluşturulmadan önce yapılan istekler gibi kimliği doğrulanmamış isteklerde de yaygın olarak kullanılır.

Ancak, yönlendiricinin kontrol edilmesi CSRF korumasının daha zayıf bir türü olarak kabul edilir. Örneğin, açık yönlendirme güvenlik açıkları, yönlendirici kontrolü ile korunan GET tabanlı istekleri istismar etmek için kullanılabilir ve bazı kuruluşlar veya tarayıcı araçları, veri koruma biçimi olarak yönlendirici başlıklarını kaldırır. Yönlendirici kontrolleriyle ilgili yaygın uygulama hataları da vardır. Örneğin, CSRF saldırısı bir HTTPS alanından kaynaklanıyorsa, yönlendirici atlanacaktır. Bu durumda, istek bir durum değişikliği gerçekleştirirken yönlendiricinin olmaması bir saldırı olarak değerlendirilmelidir. Ayrıca saldırganın yönlendirici üzerinde sınırlı etkisi olduğunu unutmayın. Örneğin, kurbanın etki alanı "site. com" ise, bir saldırgan CSRF istismarının "site.com.attacker.com" kaynaklı olmasını sağlayabilir ve bu da bozuk bir yönlendirici kontrolü uygulamasını kandırabilir. XSS, bir yönlendirici kontrolünü atlamak için kullanılabilir.

Kısacası, yönlendirici denetimi tam bir koruma olmasa da CSRF saldırılarını tespit etme ve önlemenin makul bir şeklidir. Yönlendirici denetimi bazı saldırıları tespit edebilir ancak tüm saldırıları durduramaz. Örneğin, HTTP yönlendiricisi farklı bir etki alanından geliyorsa ve yalnızca kendi etki alanınızdan gelen istekleri bekliyorsanız, bu isteği güvenle engelleyebilirsiniz.

**Menşe Üstbilgisini Kontrol Etme**
Origin HTTP Header standardı, CSRF ve diğer Cross-Domain saldırılarına karşı bir savunma yöntemi olarak tanıtılmıştır. Referanstan farklı olarak, bir HTTPS URL'sinden kaynaklanan HTTP isteğinde orijin mevcut olacaktır. Orijin başlığı mevcutsa, tutarlılık açısından kontrol edilmelidir.

**Meydan Okuma-Yanıt**
Challenge-Response CSRF için bir başka savunma seçeneğidir. Daha önce de belirtildiği gibi, genellikle çağrılan işlevsellik yüksek riskli olduğunda kullanılır. Meydan okuma-yanıt CSRF'ye karşı çok güçlü bir savunma olsa da (doğru uygulama varsayıldığında), kullanıcı deneyimini etkiler. Yüksek güvenliğe ihtiyaç duyan uygulamalar için, yüksek riskli işlevlerde belirteçler (şeffaf) ve meydan okuma yanıtı kullanılmalıdır.

**Aşağıda meydan okuma-yanıt seçeneklerine bazı örnekler verilmiştir:**
- CAPTCHA
- Yeniden Kimlik Doğrulama (şifre)
- Tek seferlik Token

**Siteler Arası Komut Dosyası (XSS) Güvenlik Açığı Yok**
Siteler Arası Komut Dosyası CSRF'nin çalışması için gerekli değildir. Ancak, herhangi bir siteler arası komut dosyası açığı token, Double-Submit cookie, referer ve origin tabanlı CSRF savunmalarını yenmek için kullanılabilir. Bunun nedeni, bir XSS yükünün bir XMLHttpRequest kullanarak sitedeki herhangi bir sayfayı okuyabilmesi ve yanıttan oluşturulan belirteci elde edebilmesi ve bu belirteci sahte bir istekle ekleyebilmesidir. Bu teknik, MySpace (Samy) solucanının 2005 yılında MySpace'in CSRF karşıtı savunmasını nasıl aştığını ve solucanın yayılmasını sağladığını göstermektedir. XSS, Captcha, yeniden kimlik doğrulama veya tek seferlik parolalar gibi meydan okuma-yanıt savunmalarını yenemez. CSRF savunmalarının aşılamadığından emin olmak için XSS güvenlik açıklarının bulunmaması zorunludur.


A9


# BILINEN GÜVENLIK AÇIKLARINA SAHIP BILEŞENLERIN KULLANILMASI

Kütüphaneler, çerçeveler ve diğer yazılım modülleri gibi bileşenler neredeyse her zaman tam ayrıcalıklarla çalışır. Güvenlik açığı olan bir bileşenden faydalanılırsa, böyle bir saldırı ciddi veri kaybını veya sunucunun ele geçirilmesini kolaylaştırabilir. Bilinen güvenlik açıklarına sahip bileşenleri kullanan uygulamalar, uygulama savunmalarını zayıflatabilir ve bir dizi olası saldırı ve etkiyi mümkün kılabilir.

## 15.1 Açıklama

Günümüzde bir uygulamanın veya yazılım bileşeninin açık kaynak kodlu veya ücretli bir kütüphane veya çerçevenin yeniden kullanımı olmadan geliştirilmesi nadirdir. Bu çerçeveler ve kütüphaneler halihazırda geliştirilmiş ve çalışır durumda olduğundan ve iyi derecede testlerden geçirildiğinden bu çok mantıklıdır. Ancak bu üçüncü parti bileşenler, bir saldırgan bileşen kodunda bir açık bulduğunda güvenlik açıklarının kaynağı da olabilir, aslında saldırgan bu açığın bileşeni kullanan herkes üzerinde işe yarayacağını bildiğinden bu açık daha da cazip hale gelir.

Bu sorun öyle bir hal almıştır ki popüler çerçeveler, kütüphaneler ve işletim sistemlerine yönelik açıklar/istismarlar yeraltı pazarlarında büyük meblağlar karşılığında satılmaktadır.

### Ne İncelenmeli

Kuruluşunuz bileşenin kodunu gözden geçirme görevini üstlenmediği sürece (açık kaynak kodlu olduğu ve kapalı kaynak kodlu bir üçüncü taraf kütüphanesi olmadığı varsayılırsa), bu konu için gerçekten gözden geçirilecek bir kod yoktur, bu durumda kod incelemesi diğer denetim incelemelerine benzer olacaktır. Ancak kod incelemesi, kuruluşun hangi üçüncü taraf kodunu kullandığını bilmesini sağlayan daha büyük şirket çapında izleme veya denetim mekanizmaları içinde kullanılabilir.

Şirketin büyüklüğü ne olursa olsun, herhangi bir güvenlik açığı tespit edildiğinde kuruluşun uyarılabilmesini sağlamak için üçüncü taraf bileşenlerin kullanımı ve bunların sürümleri izlenmelidir. Bir ya da iki ürünü olan küçük şirketler için bu takip bir elektronik tablo ya da wiki sayfası kadar kolay olabilir, ancak 100'lerce uygulaması ya da ürünü olan daha büyük şirketler için geliştiricilerin üçüncü taraf çerçeveleri ve kütüphaneleri kullanımını takip etme görevi, bu kütüphanelerin oluşturduğu risk kadar büyüktür.

Bir şirketin 20 ürünü varsa ve bu ürünlerin her biri 5 veya 6 üçüncü taraf bileşeni kullanıyorsa (örneğin Apache web sunucuları, OpenSSL kripto kütüphaneleri, regex ve DB etkileşimleri için Java kütüphaneleri, vb), bu şirkete güvenlik açıklarının gelebileceği 100'den fazla harici kaynak bırakır. Şirket aniden heartbleed tipi bir güvenlik açığı duyarsa, kendisini ve müşterilerini korumak için tepki verebilmeli ve etkilenen uygulamaları yükseltebilmeli veya başka karşı önlemler alabilmelidir.

### Bileşenlerin Kontrolü

Büyük şirketlerin üçüncü taraf güvenlik açıklarına maruz kalmalarını sınırlandırmak için kullandıkları bir yöntem de geliştiricilerin hangi kütüphaneleri kullanabileceklerini kontrol etmektir. Örneğin, geliştiricilerin kripto kütüphanesi olarak diğer seçenekler yerine OpenSSL'in belirli bir sürümünü kullanmaları gerektiğini belirtebilirler.

Bu, yönetim ve risk kontrolörlerinin piyasadaki güvenlik açıklarına karşı risk profillerini bilmelerini sağlar, eğer bouncycastle'da bir hata ortaya çıkarsa, maruz kalmadıklarını bilirler (yani, bazı geliştiriciler ürünlerden birinde bouncycastle kullanmamıştır, çünkü kullanılacak kripto kütüphaneleri listesinde yer almamaktadır). Öte yandan, OpenSSL'de bir hata varsa, tüm yumurtaları o sepetin içindedir ve derhal yükseltmeleri gerekir.

Üçüncü taraf bileşenlerin seçeneklerini sınırlandırmanın teknik zorlukları olacağı açıktır ve böyle bir politika, en yeni ve en iyi çerçeveyi kullanmak isteyen geliştiriciler arasında popüler olmayabilir, ancak bir ürünü güvence altına almanın ilk adımı, onu hangi malzemelerle yaptığınızı bilmektir.

Böyle bir politika nasıl takip edilebilir veya uygulanabilir? Bir noktada kütüphane veya çerçeve, .dll/.so şeklinde veya kaynak kod olarak kod satırına entegre edilecektir.

**Bu tür entegrasyonlar kod incelemesine tabi tutulmalıdır ve bu kod incelemesinin bir görevi olarak incelemeyi yapan kişi kontrol edebilir:**

**1.** Kütüphane, ürün paketinde kullanılabilecek bir kütüphanedir (veya belki zaten kullanılıyordur ve geliştirici farkında değildir, bu durumda inceleme reddedilmeli ve orijinal entegrasyon kullanılmalıdır)

**2.** Herhangi bir izleme veya denetim yazılımı (basit bir hesap çizelgesi bile olsa) ürünün üçüncü taraf kütüphanesini kullandığını yansıtacak şekilde güncellenir. Bu, bir güvenlik açığı ortaya çıkarsa hızlı bir şekilde düzeltme yapılmasına olanak tanır, yani ürün yamalanacaktır.

**Somunu Kırmak için Balyoz**

Gözden geçirenin son bir sorumluluğu da ihtiyaç duyulan işlevsellik için doğru üçüncü taraf kütüphanesinin kullanıldığından emin olmak olacaktır. Birçok kütüphane, kullanılmayabilecek büyük miktarda işlevsellikle birlikte gelir. Örneğin, geliştirici regex'lerini gerçekleştirmek için bir kütüphane ekliyor, ancak bu kütüphane aynı zamanda uygulama tarafından kullanılmayan / ihtiyaç duyulmayan diğer işlevleri de içeriyor mu? Bu, uygulamanın saldırı yüzeyini artırır ve bu ekstra kod bir port açtığında ve internetle iletişim kurduğunda beklenmedik davranışlara neden olabilir.

Gözden geçiren kişi çok fazla işlevsellik/kod eklendiğini düşünüyorsa, kullanılmayan işlevselliği kapatmayı önerebilir veya daha iyisi bu işlevselliği ürüne dahil etmemenin bir yolunu bulabilir (örneğin kodu çıkararak veya kullanılmayan işlevlerin asla kullanılmaması için dalları sabit kodlayarak).

OWASP projesi "OWASP Dependency Check" kütüphane kontrolü için bir otomasyon ölçüsü sağlayabilir (https://www.owasp.org/index.php/OWASP_Dependency_Check)


A10

# A10 ONAYLANMAMIŞ YÖNLENDIRMELER VE YÖNLENDIRMELER

Web uygulamaları kullanıcıları sıklıkla başka sayfalara ve web sitelerine yönlendirir ve hedef sayfaları belirlemek için güvenilmeyen veriler kullanır. Uygun doğrulama olmadan, saldırganlar kurbanları kimlik avı veya kötü amaçlı yazılım sitelerine yönlendirebilir veya yetkisiz sayfalara erişmek için yönlendirmeleri kullanabilir.

## 16.1 Açıklama

Geçersiz yönlendirmeler ve yönlendirmeler, bir web uygulaması güvenilmeyen girdiyi kabul ettiğinde mümkündür ve bu da web uygulamasının isteği güvenilmeyen girdi içinde bulunan bir URL'ye yönlendirmesine neden olabilir. Bir saldırgan, bir siteye güvenilmeyen URL girdisini değiştirerek başarılı bir kimlik avı dolandırıcılığı başlatabilir ve kullanıcı kimlik bilgilerini çalabilir.

Değiştirilen bağlantıdaki sunucu adı orijinal siteyle aynı olduğundan, kimlik avı girişimleri daha güvenilir bir görünüme sahip olabilir. Geçersiz yönlendirme ve yönlendirme saldırıları, uygulamanın erişim denetimi kontrolünü geçecek ve ardından saldırganı normalde erişemeyeceği ayrıcalıklı işlevlere yönlendirecek bir URL'yi kötü amaçlı olarak oluşturmak için de kullanılabilir.

**Yönlendirmeler**

Bir web sitesindeki yönlendirme işlevi, kullanıcının tarayıcısına sitede farklı bir sayfaya gitmesinin söylenmesini sağlar. Bu, kullanıcı arayüzünü geliştirmek veya kullanıcıların sitede nasıl gezindiğini izlemek için yapılabilir.

Yönlendirme işlevselliğini sağlamak için bir sitenin yönlendirmeyi gerçekleştirmek için belirli bir URL'si olabilir:

- **http://www.example.com/utility/redirect.cgi**

Bu sayfa 'URL' parametresini (URL'den veya POST gövdesinden) alacak ve kullanıcının tarayıcısına örneğin o sayfaya gitmesi için bir mesaj gönderecektir:

- **http://www.example.com/utility/redirect.cgi?URL=http://www.example.com/viewtxn.html**

Ancak bir saldırgan, geçerli bir kullanıcının www.example.com için gibi görünen ancak example.com'daki yönlendirme işlevini çağırarak kullanıcının tarayıcısının kötü amaçlı bir siteye (example.com gibi görünebilecek ve kullanıcıyı hassas veya kimlik doğrulama bilgilerini girmesi için kandırabilecek bir siteye) gitmesine neden olacak bir bağlantıya tıklamasını sağlamaya çalışabileceğinden bu durum kötüye kullanılabilir:

- **http://www.example.com/utiltiy/redirect cgi?URL=http://attacker.com/fakelogin.html**

**Forvetler**

Yönlendirmeler yönlendirmelere benzer ancak yeni sayfa kullanıcı tarayıcısı tarafından alınmaz (yönlendirmede olduğu gibi), bunun yerine sunucu çerçevesi yönlendirilen sayfayı alır ve kullanıcı tarayıcısına geri gönderir. Bu, Java çerçevelerindeki (örneğin Struts) 'forward' komutları veya .Net'teki 'Server.Transfer' ile gerçekleştirilir. Yönlendirme sunucu çerçevesinin kendisi tarafından gerçekleştirildiğinden, saldırganın mevcut web sitesinde kullanabileceği URL aralığını sınırlar (yani saldırgan attacker.com'a 'yönlendiremez'), ancak bu saldırı erişim kontrollerini atlamak için kullanılabilir. Örneğin, bir sitenin iletilen sayfayı yanıt olarak göndermesi gibi:

- Eğer satın alıyorsanız, 'purchase.do' adresine yönlendirin
- İptal ediliyorsa, 'cancelled.do' adresine iletin

Bu daha sonra web sitesine bir parametre olarak aktarılacaktır:

- **http://www.example.com/txn/acceptpayment.html?FWD=purchase**

Bunun yerine bir saldırgan web sitesi içinde admin.do gibi farklı bir sayfaya erişmeye çalışmak için iletme özelliğini kullanırsa, görüntüleme yetkisi olmayan sayfalara erişebilir, çünkü yetkilendirme iletilen sayfa yerine 'ödemeyi kabul et' sayfasında uygulanmaktadır.

### Ne İncelenmeli

Yönlendirilen veya yeniden yönlendirilen URL'nin herhangi bir kısmı kullanıcı girdisine dayanıyorsa, site risk altında olabilir. Emin olun:

- Tüm yönlendirmeler/yönlendirmeler bir beyaz listeye göre oluşturulur veya
- Tüm yeniden yönlendirmeler/ileri yönlendirmeler, güvenilir sitede kalmalarını sağlamak için reletive yollar kullanır

### Yönlendirmeler

Aşağıdaki örnekler güvenli olmayan yönlendirme ve yönlendirme kodunu göstermektedir. Aşağıdaki Java kodu URL'yi 'url' parametresinden alır ve bu URL'ye yönlendirir.

**Örnek 16.1**

```
response.sendRedirect(request.getParameter("url"));
```

Aşağıdaki PHP kodu sorgu dizesinden bir URL alır ve ardından kullanıcıyı bu URL'ye yönlendirir.

**Örnek 16.2**

```
$redirect_url = $_GET['url'];
 header("Location: " . $redirect_url);
```

Benzer bir C# .NET Savunmasız Kod örneği:

**Örnek 16.3**

```
string url = request.QueryString["url"];
 Response.Redirect(url);
```

Yukarıdaki kod, URL'nin kesinliğini doğrulamak için herhangi bir doğrulama veya ekstra yöntem kontrolü uygulanmazsa bir saldırıya karşı savunmasızdır. Bu güvenlik açığı, kullanıcıları zararlı bir siteye yönlendirerek kimlik avı dolandırıcılığının bir parçası olarak kullanılabilir. Kullanılacak URL'nin bir parçası olarak kullanıcı girdisinin kullanılması gerekiyorsa, girdiye sıkı bir doğrulama uygulayarak amaçlananın dışında kullanılamayacağından emin olun.

Savunmasız kodun bir 'yönlendirme' işlevini açıkça çağırmasına gerek olmadığını, bunun yerine istemci tarayıcısının yönlendirilen sayfaya gitmesine neden olmak için yanıtı doğrudan değiştirebileceğini unutmayın. Aranacak kodlar **tablo 22'**de gösterilmiştir.

**Tablo 22:** Yönlendirme

| Yeniden Yönlendirme Yöntemi | Açıklama |
|---|---|
| Yönlendirme Yanıtı (301 ve 307 yanıtlarının da yönlendirmeye neden olacağını unutmayın) | HTTP/1.1 302 Bulundu<br>Konum: http://www.attacker.com/page.html |
| Meta Etiketi | <html><head><br><meta http-equiv="Refresh"<br>content="0;url=http://attacker.com/page.html"> |
| JavaScript | <script type="text/javascript"><br><!-<br>window.location="http://attacker.com/page.html"<br>//--> |
| Başlığı Yenile | HTTP/1.1 200 TAMAM<br>Refresh=0; url=http://attacker.com/page.html |

Bir saldırganın bir forumda yönlendirme URL'si yayınladığı veya bir e-posta gönderdiği durumlarda, web sitesi kullanıcının site içindeki bir sayfadan geldiğinden emin olmak için yönlendirici başlığını kontrol edebilir, ancak kötü amaçlı URL sitenin kendi içinde yer alıyorsa bu karşı önlem geçerli olmayacaktır.

Yönlendirmenin gitmesine izin verilen URL'lerin veya seçeneklerin bir beyaz listesini oluşturmayı düşünün veya kullanıcı girdisinin yönlendirmenin şemasını veya ana bilgisayar adını belirleme yeteneğini reddedin. Bir site ayrıca, bir saldırganın kodlanmadığında (veya şifrelenmediğinde) geçerli kabul edilecek kötü amaçlı bir URL parametresini kolayca oluşturamayacağı şekilde yönlendirilecek URL değerini kodlayabilir (veya şifreleyebilir).

**Forvetler**

Yönlendirmeler için karşı önlem, yönlendirilebilecek sayfa aralığını beyaz listeye almak (yönlendirmelere benzer şekilde) ve yönlendirilen sayfanın yanı sıra yönlendirme sayfasında da kimlik doğrulamasını zorunlu kılmaktır. Bu, bir saldırganın erişimi olmaması gereken bir sayfaya yönlendirmeyi zorlamayı başarsa bile, yönlendirilen sayfadaki kimlik doğrulama kontrolünün erişimi reddedeceği anlamına gelir.

**J2EE Hakkında Not**

J2EE uygulamalarında "sendRedirect" yöntemiyle ilgili bir hata olduğu belirtilmektedir. Örneğin:

- response.sendRedirect("home.html");

Bu yöntem, kullanıcıya bir yeniden yönlendirme yanıtı göndermek için kullanılır ve kullanıcı daha sonra URL'si yönteme bir argüman olarak aktarılan istenen web bileşenine yönlendirilir. Bu tür bir yanlış anlama, kullanıcıyı yönlendiren Servlet/JSP sayfasındaki yürütme akışının bu yöntemin çağrılmasından sonra durmasıdır. 'If' koşulundan sonra kod varsa bunun yürütüleceğini unutmayın.

Bir servlet ya da JSP'nin sendRedirect() metodundan sonra bile çalışmasının devam etmesi, RequestDispatcher sınıfının Forward metodu için de geçerlidir. Ancak <jsp:forward> etiketi bir istisnadır, <jsp:forward> etiketinin kullanımından sonra yürütme akışının durduğu gözlemlenmektedir.

Yönlendirme veya yönlendirme yaptıktan sonra, bir "return" deyimi kullanarak kod akışını sonlandırın.

**Referanslar**

- Açık Yönlendirmeler Hakkında OWASP Makalesi **https://www.owasp.org/index.php/Open_redirect**

- Açık Yönlendirmelerde CWE Girişi 601 **http://cwe.mitre.org/data/definitions/601.html**

- URL Yönlendiricisinin Kötüye Kullanımı Hakkında WASC Makalesi **http://projects.webappsec.org/w/page/13246981/URL%20Redi- rector%20Abuse**

- Açık yönlendirmelerin tehlikeleri hakkında Google blog makalesi **http://googlewebmastercentral.blogspot. com/2009/01/open-redirect-urls-is-your-site-being.html**

- Açık Yönlendirme Saldırılarını Önleme (C#) **http://www.asp.net/mvc/tutorials/security/preventing-open-re- direction-attacks**

# HTML5

HTML5, HTLML4, XHTML ve HTML DOM Seviye 2'nin yerini almak üzere oluşturulmuştur. Bu yeni standardın temel amacı, ekstra tescilli istemci tarafı eklentileri kullanmadan dinamik içerik sağlamaktır. Bu, tasarımcıların ve geliştiricilerin tarayıcıya herhangi bir eklenti yüklemek zorunda kalmadan harika bir kullanıcı deneyimi sağlayan olağanüstü siteler oluşturmalarına olanak tanır.

## 17.1 Açıklama

İdeal olan kullanıcıların en yeni web tarayıcısını yüklemiş olmalarıdır, ancak bu güvenlik uzmanlarının tavsiye ettiği kadar düzenli bir şekilde gerçekleşmemektedir, bu nedenle web sitesi 2 katmanlı kontroller uygulamalıdır, bir katman tarayıcı türünden bağımsız, ikincisi ise ek bir kontrol olarak.

**Ne İncelenmeli: Web Mesajlaşma**

Web Mesajlaşması (Çapraz Etki Alanı Mesajlaşması olarak da bilinir), geçmişte bu görevi yerine getirmek için kullanılan çoklu hack'lerden genellikle daha güvenli bir şekilde farklı kökenlerden gelen belgeler arasında mesajlaşma aracı sağlar. İletişim API'si aşağıdaki gibidir:

**Ancak yine de akılda tutulması gereken bazı öneriler vardır:**

• Bir mesaj gönderirken, bir yönlendirme veya hedef pencerenin kaynağının başka bir şekilde değiştirilmesinden sonra mesajın bilinmeyen bir kaynağa gönderilmesini önlemek için beklenen kaynağı '*' yerine 'postMessage' için ikinci argüman olarak açıkça belirtin.

• Alıcı sayfa her zaman olmalıdır:

o Verilerin beklenen konumdan geldiğini doğrulamak için gönderenin 'origin' özelliğini kontrol edin.
o İstenilen formatta olduğundan emin olmak için olayın 'data' niteliği üzerinde girdi doğrulaması gerçekleştirin.

• 'data' özelliği üzerinde kontrol sahibi olduğunuzu varsaymayın. Gönderme sayfasındaki tek bir Siteler Arası Komut Dosyası açığı, bir saldırganın herhangi bir formatta mesaj göndermesine olanak tanır.

• Her iki sayfa da değiş tokuş edilen mesajları yalnızca 'veri' olarak yorumlamalıdır. İletilen mesajları asla kod olarak değerlendirmeyin (örn. 'eval()' ile) veya bir sayfa DOM'una eklemeyin (örn. 'innerHTML' ile), çünkü bu DOM tabanlı bir XSS güvenlik açığı yaratacaktır.

• Veri değerini bir öğeye atamak için 'element.innerHTML = data' gibi güvensiz bir yöntem kullanmak yerine daha güvenli olan 'element.textContent = data;' seçeneğini kullanın.

• Beklediğiniz FQDN(ler) ile eşleşmesi için orijini tam olarak kontrol edin. Aşağıdaki koda dikkat edin: ' if(message. orgin.indexOf(".owasp.org")!=-1) { /* ... */ }' çok güvensizdir ve 'www.owasp. org.attacker.com' eşleşeceği için istenen davranışa sahip olmayacaktır.

• Harici içerik/güvenilmeyen araçlar yerleştirmeniz ve kullanıcı kontrollü komut dosyalarına izin vermeniz gerekiyorsa (ki bu kesinlikle önerilmez), Google'ın Caja'sı gibi bir JavaScript yeniden yazma çerçevesi kullanmayı düşünün veya korumalı çerçeveler hakkındaki bilgileri kontrol edin.

**Ne İncelenmeli? Çapraz Menşe Kaynak Paylaşımı**
Çapraz Kaynak Paylaşımı veya CORS, bir web tarayıcısının XMLHttpRequest L2 API'sini kullanarak kontrollü bir şekilde "çapraz alan" istekleri gerçekleştirmesini sağlayan bir mekanizmadır. Geçmişte XMLHttpRequest L1 API'si, aynı menşe politikası tarafından kısıtlandığı için isteklerin yalnızca aynı menşe içinde gönderilmesine izin veriyordu.

Çapraz Orijinli isteklerde, isteği başlatan etki alanını tanımlayan ve tarayıcı tarafından sunucuya gönderilen isteğe otomatik olarak dahil edilen bir Orijin başlığı bulunur. CORS, bir web tarayıcısı ile bir sunucu arasında, çapraz kökenli bir isteğe izin verilip verilmediğini belirleyecek protokolü tanımlar. Bu hedefe ulaşmak için, mesajlaşma bağlamı hakkında bilgi sağlayan HTTP başlıkları vardır: Köken, Erişim-Kontrol-İstek-Metodu, Erişim-Kontrol-İstek-Başlıkları, Erişim-Kontrol-İzin Ver-Köken, Erişim-Kontrol-İzin Ver-Kimlik Bilgileri, Erişim-Kontrol-İzin Ver-Metotları, Erişim-Kontrol-İzin Ver-Başlıkları.

CORS spesifikasyonu, GET veya POST dışındaki istekler veya kimlik bilgileri kullanan istekler gibi basit olmayan istekler için, istek türünün veriler üzerinde olumsuz bir etkisi olup olmayacağını kontrol etmek için önceden bir uçuş öncesi OPTIONS isteği gönderilmesini zorunlu kılar. Uçuş öncesi istek, sunucu tarafından izin verilen yöntemleri, başlıkları kontrol eder ve kimlik bilgilerine izin veriliyorsa, OPTIONS isteğinin sonucuna göre tarayıcı isteğe izin verilip verilmeyeceğine karar verir.

**CORS ile ilgili kodu incelerken dikkat edilmesi gerekenler şunlardır:**
• 'Access-Control-Allow-Origin: '*' ile yanıt veren URL'lerin saldırganın daha sonraki saldırılarına yardımcı olabilecek herhangi bir hassas içerik veya bilgi içermediğinden emin olun. 'Access-Control-Allow-Origin' başlığını yalnızca etki alanları arasında erişilmesi gereken seçilmiş URL'lerde kullanın. Bu başlığı tüm etki alanı için kullanmayın.

• 'Access-Control-Allow-Origin' başlığında yalnızca seçili, güvenilir alan adlarına izin verin. Kara listeye almak veya herhangi bir etki alanına izin vermek yerine etki alanlarını beyaz listeye almayı tercih edin ('*' joker karakterini kullanmayın veya 'Origin' başlık içeriğini herhangi bir kontrol yapmadan körü körüne döndürmeyin).

• CORS'un istenen verilerin kimliği doğrulanmamış bir konuma gitmesini engellemediğini unutmayın. Sunucunun olağan Siteler Arası İstek Sahteciliği önleme işlemini gerçekleştirmesi hala önemlidir.

• RFC, 'OPTIONS' fiili ile uçuş öncesi bir istek önerirken, mevcut uygulamalar bu isteği algılamayabilir, bu nedenle "sıradan" ('GET' ve 'POST') isteklerin gerekli erişim kontrolünü gerçekleştirmesi önemlidir.

• Karışık içerik hatalarını önlemek için HTTPS kökenli düz HTTP üzerinden alınan istekleri atın.

• Erişim Kontrolü kontrolleri için yalnızca Origin başlığına güvenmeyin. Tarayıcı CORS isteklerinde her zaman bu başlığı gönderir, ancak tarayıcı dışında yanıltılabilir. Hassas verileri korumak için uygulama düzeyinde protokoller kullanılmalıdır.

**Ne İncelenmeli? WebSockets**
Geleneksel olarak HTTP protokolü TCP bağlantısı başına yalnızca bir istek/yanıt yapılmasına izin verir. Asenkron JavaS-cript ve XML (AJAX) istemcilerin sunucuya asenkron olarak (sayfa yenileme olmadan arka planda) veri göndermesine ve almasına izin verir, ancak AJAX istemcinin istekleri başlatmasını ve sunucu yanıtlarını beklemesini gerektirir (yarı-dubleks). HTML5 WebSockets, istemci/sunucunun 'tam çift yönlü' (iki yönlü) iletişim kanalları oluşturmasına olanak tanıyarak istemci ve sunucunun gerçekten eşzamansız iletişim kurmasını sağlar. WebSockets ilk 'yükseltme' el sıkışmasını HTTP üzerinden gerçekleştirir ve bundan sonra tüm iletişim TCP kanalları üzerinden yürütülür.

**Aşağıda Web Sockets kullanan bir uygulamanın örnek kodu verilmiştir:**

## Örnek 17.1

```
[Kurucu(in DOMString url, isteğe bağlı in DOMString protokol)]
 arayüz WebSocket
 { readonly attribute DOMString URL;
 // ready state const unsigned short CONNECTING = 0;
 const unsigned short OPEN = 1;
 const unsigned short CLOSED = 2;
 readonly attribute unsigned short readyState;
 readonly attribute unsigned long bufferedAmount;
 // ağ oluşturma
 attribute Function onopen;
 attribute Function onmessage;
 attribute Function onclose;
 boolean send(in DOMString data);
 void close();
 };
 WebSocket EventTarget uygular;

 var myWebSocket = new WebSocket("ws://www.websockets.org");

 myWebSocket.onopen = function(evt) { alert("Bağlantı açık..."); }; myWebSocket.onmessage
 = function(evt) { alert("Alınan Mesaj: " + evt.data); }; myWebSocket.onclose = function(evt) {
 alert("Bağlantı kapalı."); };
```

**Web soketlerini uygulayan kodu incelerken aşağıdaki öğeler göz önünde bulundurulmalıdır:**

• Uygulanan istemci/sunucularda geriye dönük uyumluluğu bırakın ve yalnızca hybi-00'ün üzerindeki protokol sürümlerini kullanın. Popüler Hixie-76 sürümü (hiby-00) ve daha eski sürümler eski ve güvensizdir.

• Tüm güncel tarayıcıların en son sürümlerinde desteklenen önerilen sürüm rfc6455 RFC 6455'tir (Firefox 11+, Chrome 16+, Safari 6, Opera 12.50 ve IE10 tarafından desteklenmektedir).

• TCP hizmetlerini WebSocket'ler aracılığıyla tünellemek nispeten kolay olsa da (örneğin VNC, FTP), bunu yapmak, bir Çapraz Site Komut Dosyası saldırısı durumunda tarayıcı içi saldırgan için bu tünellenmiş hizmetlere erişim sağlar. Bu hizmetler doğrudan kötü niyetli bir sayfa veya programdan da çağrılabilir.

• Protokol yetkilendirme ve/veya kimlik doğrulama işlemlerini gerçekleştirmez. Uygulama düzeyindeki protokoller, hassas verilerin aktarılması durumunda bunu ayrı olarak ele almalıdır.
• Websocket tarafından alınan mesajları veri olarak işleyin. Doğrudan DOM'a atamaya çalışmayın veya kod olarak değerlendirmeyin. Yanıt JSON ise, asla güvenli olmayan eval() işlevini kullanmayın; bunun yerine güvenli seçenek olan JSON.parse() işlevini kullanın.

• 'ws://'protokolü aracılığıyla açığa çıkan uç noktalar kolayca düz metne dönüştürülebilir. Man-In-The-Middle saldırılarına karşı koruma için yalnızca'wss://' (SSL/ TLS üzerinden WebSockets) kullanılmalıdır.

• Tarayıcı dışında istemciyi yanıltmak mümkündür, bu nedenle WebSockets sunucusu yanlış/ kötü niyetli girdileri işleyebilmelidir. Değiştirilmiş olabileceğinden, uzak siteden gelen girdiyi her zaman doğrulayın.
• Sunucuları uygularken, Websockets el sıkışmasındaki 'Origin:' başlığını kontrol edin. Sahte olsa da

dışında, tarayıcılar her zaman Websockets bağlantısını başlatan sayfanın Menşeini ekler.

• Bir tarayıcıdaki WebSockets istemcisine JavaScript çağrıları yoluyla erişilebildiğinden, tüm Websockets iletişimi Sahte veya Siteler Arası Komut Dosyası yoluyla ele geçirilebilir. Bir WebSockets bağlantısı üzerinden gelen verileri her zaman doğrulayın.

### 5.1.5 Gözden Geçirilecekler: Sunucu Tarafından Gönderilen Olaylar

Sunucu tarafından gönderilen olaylar WebSocket'lere benzer, ancak özel bir protokol kullanmazlar (HTTP'yi yeniden kullanmışlardır) ve istemci tarayıcının yalnızca sunucudan gelen güncellemeleri (mesajları) dinlemesine izin verirler, böylece istemcinin sunucuya herhangi bir yoklama veya diğer mesajları gönderme ihtiyacını ortadan kaldırırlar.

**Sunucu tarafından gönderilen olayları işleyen kodu incelerken, akılda tutulması gereken öğeler şunlardır:**

• Yalnızca aynı kaynaklı URL'lere izin verilmesine rağmen, 'EventSource' yapıcısına aktarılan URL'leri doğrulayın.

• Daha önce de belirtildiği gibi, mesajları ('event.data') veri olarak işleyin ve içeriği asla HTML veya komut dosyası kodu olarak değerlendirmeyin.

• İletinin güvenilir bir etki alanından geldiğinden emin olmak için her zaman iletinin kaynak özniteliğini ('event.origin') kontrol edin. Beyaz liste kullanın.

# AYNI MENŞE POLITIKASI

Tek Köken Politikası olarak da adlandırılan Aynı Köken Politikası (SOP), web uygulaması güvenlik modelinin bir parçasıdır. Same Origin Policy, kodu gözden geçiren kişinin dikkate alması gereken güvenlik açıklarına sahiptir. SOP, web geliştirmenin üç ana alanını kapsar: Güven, Yetki ve Politika. Same Origin Policy üç bileşenin (Scheme, Hostname ve Port) birleşiminden oluşur.

## 18.1 Açıklama

**Aynı kaynak ilkesi söz konusu olduğunda Internet Explorer'ın iki önemli istisnası vardır:**

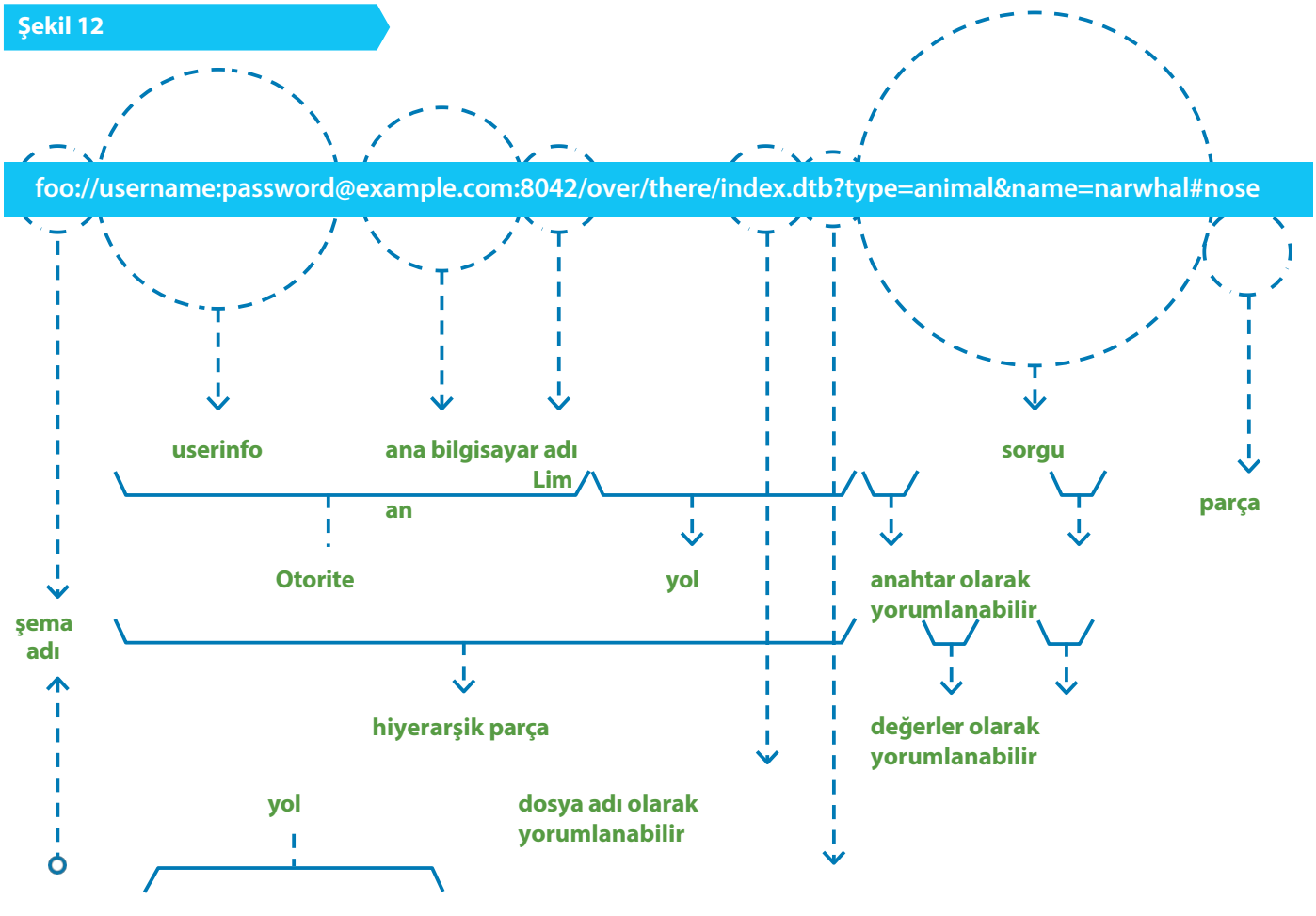
**1.** Güven Bölgeleri: Her iki etki alanı da yüksek güvenilen bölgedeyse, örneğin kurumsal etki alanları, o zaman aynı kaynak sınırlamaları uygulanmaz.

**2.** Bağlantı Noktası: IE, bağlantı noktasını Aynı Kaynak bileşenlerine dahil etmez, bu nedenle http://yourcompany.com:81/index. html ve http://yourcompany.com/index.html aynı kaynaktan kabul edilir ve herhangi bir kısıtlama uygulanmaz.

Bu istisnalar standart değildir ve diğer tarayıcıların hiçbirinde desteklenmez, ancak Windows RT (veya) IE tabanlı web uygulaması için bir uygulama geliştiriyorsanız yardımcı olacaktır.

**Aşağıdaki şekilde URL'nin çeşitli bölümleri gösterilmektedir:**

**Şekil 12**



dosya adı olarak yorumlanabilir

- foo://username:password@example.com:8042/over/there/index.dtb?type=animal&name =narwhal#nose

### 18.2 Ne İncelenmeli

- Uygulama URL'de kullanıcı tarafından sağlanan verilere izin veriyorsa, kodu gözden geçirenin yol, sorgu veya Fragment Id Code verilerinin doğrulandığından emin olması gerekir.

- Kullanıcı tarafından sağlanan şema adı veya yetki bölümünün iyi bir girdi doğrulamasına sahip olduğundan emin olun. Bu önemli bir kod ekleme ve kimlik avı riskidir. Yalnızca uygulama tarafından ihtiyaç duyulan öneklere izin verin. Kara liste kullanmayın. Kod yeniden görüntüleyici, doğrulama için yalnızca beyaz listenin kullanıldığından emin olmalıdır.

- Yetki bölümünün yalnızca alfanümerikler, "-" ve "." içerdiğinden emin olun. Ve ardından "/", "?", "#" gelmelidir. Buradaki risk bir IDN homograf saldırısıdır.

- Kodu gözden geçiren kişinin, programcının varsayılan davranışı varsaymadığından emin olması gerekir çünkü programcının tarayıcısı belirli bir karakteri düzgün bir şekilde kaçar veya tarayıcı standardı, URL'den türetilen herhangi bir değerin bir veritabanı sorgusuna yerleştirilmesine veya URL'nin kullanıcıya geri yankılanmasına izin vermeden önce karakterin düzgün bir şekilde kaçacağını söyler.

- MIME türü image/png olan kaynaklar resim olarak, MIME türü text/html olan kaynaklar ise HTML belgesi olarak değerlendirilir. Web uygulamaları, MIME türünü kısıtlayarak bu içeriğin yetkisini sınırlandırabilir. Örneğin, kullanıcı tarafından oluşturulan içeriği image/png olarak sunmak, kullanıcı tarafından oluşturulan içeriği text/html olarak sunmaktan daha az risklidir.

- Belge ve kaynaklar üzerindeki ayrıcalıklar, bir bütün olarak menşeilere ayrıcalıklar vermeli veya vermemelidir (bir menşe içindeki münferit belgeler arasında ayrım yapmak yerine). Ayrıcalıkların esirgenmesi etkisizdir çünkü ayrıcalığa sahip olmayan belge genellikle ayrıcalığı yine de elde edebilir çünkü SOP bir menşe içindeki belgeleri izole etmez.

# GÜNLÜK KODUNUN GÖZDEN GEÇIRILMESI

Uygulamalar farklı yoğunluktaki mesajları farklı alıcılara günlüğe kaydeder. Birçok günlük API'si, günlük mesajının ayrıntı düzeyini neredeyse tüm mesajların 'izleme' veya 'hata ayıklama' düzeyinde günlüğe kaydedilmesinden yalnızca en önemli mesajların 'kritik' düzeyde günlüğe kaydedilmesine kadar ayarlamanıza olanak tanır. Günlük mesajının nereye yazılacağı da önemli bir husustur; bazen yerel bir dosyaya, bazen bir veritabanı günlük tablosuna yazılabilir ya da bir ağ bağlantısı üzerinden merkezi bir günlük sunucusuna yazılabilir.

Günlüğe mesaj yazma eylemi CPU döngülerini kullandığından, günlüğe her küçük ayrıntıyı yazmak daha fazla kaynak (CPU, ağ bant genişliği, disk alanı) kullanacağından günlük hacminin kontrol edilmesi gerekir. Günlüklerin kullanılabilir olması için bir araç veya insan tarafından ayrıştırılması veya yorumlanması gerektiği gerçeğiyle birleştiğinde, günlüğün tüketicisi önemli bir mesajı bulmak için binlerce satırı ayrıştırmak zorunda kalabilir.

## 19.1 Açıklama

Günlükler türüne göre değişebilir; örneğin bir günlük sadece uygulama durumunu veya süreç verilerini içerebilir, böylece bir hata oluştuğunda sistemin ne yaptığının destek portu veya geliştirme tarafından takip edilmesini sağlar. Diğer günlükler güvenliğe özel olabilir, yalnızca merkezi bir güvenlik sisteminin ilgileneceği önemli bilgileri günlüğe kaydeder. Diğer günlükler faturalandırma gibi iş amaçları için kullanılabilir.

Uygulama iş mantığı kullanıcı (örneğin kimlik, roller, izinler) ve olayın bağlamı (hedef, eylem, sonuçlar) hakkında en fazla bilgiye sahip olduğu için uygulama günlükleri güçlü olabilir ve genellikle bu veriler altyapı cihazlarında veya hatta yakından ilişkili uygulamalarda mevcut değildir. Uygulama günlüğü, özellikle destek personeli ve denetçiler için bir üretim sisteminin önemli bir özelliğidir, ancak genellikle unutulur ve tasarım / gereksinim belgelerinde nadiren yeterli ayrıntıda açıklanır. Güvenlik izleme, uyarı ve raporlama düzeyi ve kapsamı projelerin gereksinim ve tasarım aşamasında belirlenmeli ve bilgi güvenliği riskleriyle orantılı olmalıdır. Uygulama günlük kaydı da uygulama içinde tutarlı olmalı, bir kuruluşun uygulama portföyü genelinde tutarlı olmalı ve ilgili olduğunda endüstri standartlarını kullanmalıdır, böylece günlüğe kaydedilen olay verileri çok çeşitli sistemler tarafından tüketilebilir, ilişkilendirilebilir, analiz edilebilir ve yönetilebilir.

Her tür uygulama, olay verilerini doğrudan bir ağ bağlantısı üzerinden veya günlüklerin günlük/haftalık/aylık güvenli bir kopyası aracılığıyla merkezi bir günlük toplama ve yönetme sistemine (örneğin SIEM veya SEM) veya başka bir yerdeki başka bir uygulamaya asenkron olarak uzak sistemlere gönderebilir.

Günlükteki bilgiler önemliyse ve muhtemelen yasal konularda kullanılabilecekse, kaynağın (günlüğün) nasıl doğrulanabileceğini ve bütünlüğün ve inkar etmemenin nasıl sağlanabileceğini düşünün. Günlük verileri, geçici hata ayıklama günlükleri ve yedekler/kopyalar/çıktılar, gerekli veri saklama süresinden önce imha edilmemeli ve bu sürenin ötesinde saklanmamalıdır. Yasal, düzenleyici ve sözleşmesel yükümlülükler bu süreleri etkileyebilir.

Sunucu uygulamaları genellikle olay günlüğü verilerini dosya sistemine veya bir veritabanına (SQL veya NoSQL) yazar, ancak masaüstü bilgisayarlara ve mobil cihazlara yüklenen uygulamalar gibi istemci cihazlarda günlük kaydı gerekebilir ve yerel depolama ve yerel veritabanları kullanabilir. Bu istemci günlük verilerinin sunucuya nasıl aktarıldığını düşünün.

**Ne İncelenmeli**

Kod modüllerini günlük kaydı açısından incelerken, dikkat edilmesi gereken bazı yaygın sorunlar şunlardır:

• Dosya sistemini kullanırken, işletim sistemi, diğer uygulama dosyaları ve kullanıcı tarafından oluşturulan içerik tarafından kullanılanlardan ayrı bir bölüm kullanılması tercih edilir

• Dosya tabanlı günlükler için, dizinlere hangi kullanıcıların erişebileceğine ve dizinlerdeki dosyaların izinlerine ilişkin katı izinler uygulayın
• Web uygulamalarında, günlükler web üzerinden erişilebilen konumlarda gösterilmemeli, gösterilecekse de erişimi kısıtlı olmalı ve düz metin MIME türüyle (HTML değil) yapılandırılmalıdır

• Bir veritabanı kullanırken, yalnızca günlük verilerini yazmak için kullanılan ve çok kısıtlı veritabanı, tablo, işlev ve komut izinlerine sahip ayrı bir veritabanı hesabı kullanmak tercih edilir

• Ne tür mesajların kaydedilmesi gerektiğini düşünün:
o Protokol ihlalleri, kabul edilemez kodlamalar, geçersiz parametre adları ve değerleri gibi girdi doğrulama hataları

o Çıktı doğrulama hataları, örneğin veritabanı kayıt seti uyuşmazlığı, geçersiz veri kodlaması

o Kimlik doğrulama başarıları ve başarısızlıkları

o Yetkilendirme (erişim kontrolü) hataları

o Oturum yönetimi hataları, örneğin çerez oturum tanımlama değerinin değiştirilmesi

o Bağlantı zamanlamaları

• Her bir günlük mesajının ne içermesi gerektiğini düşünün:
o Tarih ve saat, bazı ortak formatlarda (bir uygulamanın tüm düğümlerinin NTP gibi bir şey aracılığıyla senkronize edilmesini sağlamak da mantıklıdır

o Eylemi gerçekleştiren kullanıcı

o Gerçekleştirilen/denenen eylem

o İstemci hakkında bilgi, örneğin IP adresi, kaynak bağlantı noktası, kullanıcı aracısı

o Harici sınıflandırmalar, örneğin NIST Güvenlik İçeriği Otomasyon Protokolü (SCAP), Mitre Ortak Saldırı Paterni Numaralandırma ve Sınıflandırma (CAPEC)

o Günlük ekleme saldırılarını önlemek için tüm olay verilerinde sanitizasyon gerçekleştirin, örneğin satır başı (CR), satır besleme (LF) ve sınırlayıcı karakterler (ve isteğe bağlı olarak hassas verileri kaldırmak için)

• Veritabanlarına yazıyorsanız, SQL enjeksiyonu hile sayfasını okuyun, anlayın ve uygulayın

• Uygulama güvenliği, fuzz, sızma ve performans testleri sırasında günlük kaydının uygulandığından ve etkinleştirildiğinden emin olun

• Günlük kaydının, örneğin disk alanını doldurarak veya veritabanı işlem günlüğü alanını aşarak sistem kaynaklarını tüketmek ve hizmet reddine yol açmak için kullanılamayacağından emin olun

• Kayıt mekanizmaları ve toplanan olay verileri, aktarım sırasında kurcalama ve depolandıktan sonra yetkisiz erişim, değişiklik ve silme gibi yanlış kullanımlara karşı korunmalıdır

• Günlük verilerini mümkün olan en kısa sürede salt okunur ortama depolayın veya kopyalayın

• Neyin kaydedilmemesi gerektiğini düşünün:
o Oturum tanımlama değerleri (oturuma özgü olayları izlemek için gerekirse karma bir değerle değiştirmeyi düşünün)

o Hassas kişisel veriler ve bazı kişisel olarak tanımlanabilir bilgi türleri (PII)

o Kimlik doğrulama parolaları (başarılı veya başarısız)

o Veritabanı bağlantı dizeleri

o Anahtarlar

o Kayıt sisteminin depolamasına izin verilenden daha yüksek bir güvenlik sınıflandırmasına sahip veriler

**Referanslar**

• Daha fazla rehberlik için NIST SP 800-92 Bilgisayar Güvenliği Günlük Yönetimi Kılavuzu'na bakın.

• Mitre Ortak Olay İfadesi (CEE)

• PCISSC PCI DSS v2.0 Gereksinim 10 ve PA-DSS v2.0 Gereksinim 4

• Diğer Ortak Günlük Dosya Sistemi (CLFS), Microsoft

# HATA İŞLEME

**Doğru hata işleme iki açıdan önemlidir:**
**1.** Uygulamanın durumunu etkileyebilir. Hatayı önlemedeki ilk başarısızlık, uygulamanın güvensiz bir duruma geçmesine neden olabilir. Bu, 'güvenli bir şekilde başarısız olmanın' temel önermesini kapsar, neden olunan hatalar uygulamayı güvensiz bir durumda bırakmamalıdır. Kaynaklar kilitlenmeli ve serbest bırakılmalı, oturumlar sonlandırılmalı (gerekirse) ve hesaplamalar veya iş mantığı durdurulmalıdır (elbette hatanın türüne bağlı olarak).

**2.** Sistem bilgilerini bir kullanıcıya sızdırabilir. Güvenli uygulama geliştirmenin önemli bir yönü bilgi sızıntısını önlemektir. Hata mesajları, saldırganlara bir uygulamanın iç işleyişi hakkında büyük bilgiler verir. Zayıf hata işleme de saldırganlara yardımcı olur, çünkü döndürülen hatalar doğru saldırı vektörleri oluşturmalarına yardımcı olabilir.

Çoğu hata için genel bir hata sayfası önerilir, bu yaklaşım saldırganların "<referans içermelidir>" booleanizasyonunu kullanarak kör SQL enjeksiyonu veya yanıt süresi özelliklerinin analizi gibi potansiyel olarak başarılı saldırıların imzalarını belirlemesini zorlaştırır.

## 20.1 Açıklama

Hata İşleme kodunun gözden geçirilmesinin amacı, uygulamanın beklenen ve beklenmeyen tüm olası hata koşulları altında güvenli bir şekilde başarısız olmasını sağlamaktır. Bir hata oluştuğunda kullanıcıya hiçbir hassas bilgi sunulmaz. Bir şirketin kodlama yönergeleri, Hata İşleme ve bunun bir uygulama paketi tarafından nasıl kontrol edilmesi gerektiğine dair bölümler içermelidir; bu, geliştiricilerin bu yönergelere göre kodlama yapmasına ve bunlara karşı inceleme yapmasına olanak tanıyacaktır.

Örneğin, SQL enjeksiyonunun bazı sağlıklı hata mesajları olmadan başarılı bir şekilde yürütülmesi çok daha zordur. Saldırı ayak izini azaltır ve bir saldırgan daha zor ve zaman alıcı olan "kör SQL enjeksiyonu" kullanmaya başvurmak zorunda kalır.

**Bir şirkette iyi planlanmış bir hata/istisna işleme kılavuzu üç nedenden dolayı önemlidir:**
**1.** İyi bir hata işleme, saldırgana, uygulamaya saldırmak için bir araç olan herhangi bir bilgi vermez

**2.** Uygun bir merkezi hata stratejisinin sürdürülmesi daha kolaydır ve yakalanmamış hataların bir uygulamanın ön ucuna "kabarcıklanma" olasılığını azaltır.

**3.** Bilgi sızıntısı sosyal mühendislik istismarlarına yol açabilir, örneğin barındırma şirketlerinin adı döndürülürse veya bazı çalışanların adı görülebilirse.

**Geliştirme dilinin kontrol edilmiş istisnalar sağlayıp sağlamadığına bakılmaksızın, gözden geçirenler şunu unutmamalıdır:**
• Tüm hatalar istisna değildir. Hataları ele almanın tek yolu olarak istisnaları ele almaya güvenmeyin, tüm case deyimlerinin 'varsayılan' bölümlerini ele alın, tüm 'if' deyimlerinin 'else' cümlelerini kapsadığından emin olun, bir fonksiyondan tüm çıkışların (örn. return deyimleri, istisnalar, vb.) kapsandığından emin olun. RAII kavramları (örneğin otomatik işaretçiler ve benzerleri) burada bir avantajdır. Java ve C# gibi dillerde, hataların istisnalardan farklı olduğunu (farklı hiyerarşi) ve ele alınması gerektiğini unutmayın.

• Bir istisnayı yakalamak, onu otomatik olarak ele almak anlamına gelmez. İstisnayı yakaladınız, peki nasıl ele alacaksınız? Birçok durumda, iş mantığınıza bağlı olarak bu yeterince açık olmalıdır, ancak bazıları için (örneğin bellek dışı, dizi dizini sınır dışı vb.) işleme o kadar basit olmayabilir.

• Başa çıkabileceğinizden fazlasını yakalamayın. Tümünü yakala cümleciklerinden (örneğin Java ve C#'ta 'catch(Exception e)' veya C++'ta 'catch(...)) kaçınılmalıdır çünkü ne tür bir istisnayı ele aldığınızı bilemezsiniz ve istisna türünü bilmiyorsanız, bunu nasıl doğru bir şekilde ele alırsınız? Aşağı akış sunucusu yanıt vermiyor olabilir veya bir kullanıcı kotasını aşmış olabilir ya da belleğiniz tükenmiş olabilir, bu sorunlar farklı şekillerde ele alınmalı ve bu nedenle belirli istisna cümlelerinde yakalanmalıdır.

Bir istisna veya hata oluştuğunda, bu olayı da günlüğe kaydetmemiz gerekir. Bu bazen kötü geliştirmeden kaynaklanır, ancak bir saldırının veya uygulamanızın güvendiği başka bir hizmetin başarısız olmasının sonucu da olabilir. Bunun üretim senaryosunda hayal edilmesi gerekir, eğer uygulamanız müşteriye bir hata yanıtı döndürerek 'güvenli bir şekilde başarısız olmayı' ele alıyorsa ve bu hatanın genel olacağı bilgisini sızdırmak istemediğimiz için, hatanın neden oluştuğunu belirlemenin bir yoluna sahip olmamız gerekir. Müşteriniz dün gece 1000'lerce hata oluştuğunu bildirirse, müşterinin nedenini bilmek isteyeceğini bilirsiniz. Uygulamanızda uygun bir günlük kaydı ve izlenebilirlik kodu yoksa, bu hataların bir hack girişiminden mi yoksa belirli bir hata türünü işlerken iş mantığınızdaki bir hatadan mı kaynaklandığını belirleyemezsiniz.

Bir istisnanın fırlatılmasına neden olabilecek tüm kod yolları, istisnanın fırlatılmaması için başarıyı kontrol etmelidir. Bu, özellikle büyük kod kütleleri için manuel bir kod incelemesinin kapsaması zor veya imkansız olabilir. Ancak kodun bir hata ayıklama sürümü varsa, modüller/fonksiyonlar ilgili istisnaları/hataları atabilir ve otomatik bir araç modülden gelen durum ve hata yanıtlarının beklendiği gibi olduğundan emin olabilir. Bu durumda kod gözden geçiren kişinin görevi, ilgili tüm istisnaların/hataların hata ayıklama kodunda test edilmesini sağlamaktır.

### Ne İncelenmeli

Kodu gözden geçirirken, hata/istisna işleme perspektifinden uygulama içindeki ortaklığı değerlendirmeniz önerilir. Çerçeveler, güvenli programlamaya yardımcı olmak için kullanılabilecek hata işleme kaynaklarına sahiptir ve hata işlemenin doğru bir şekilde "bağlanıp bağlanmadığını" değerlendirmek için çerçeve içindeki bu tür kaynaklar gözden geçirilmelidir. Mümkünse tüm istisnalar için genel bir hata sayfası kullanılmalıdır çünkü bu, saldırganın hata durumlarına verilen dahili yanıtları tanımlamasını engeller. Bu aynı zamanda otomatik araçların başarılı saldırıları tespit etmesini de zorlaştırır.

JSP struts için bu, kablolanmış struts ortamını gözden geçirirken anahtar bir dosya olan struts-config.xml dosyasında kontrol edilebilir:

**Örnek 20.1**

```
<exception key="bank.error.nowonga"
        path="/NoWonga.jsp"
        type="mybank.account.NoCashException"/>
```

İşlenmemiş istisnaları ele almak için web.xml'de JSP için belirtim yapılabilir. İşlenmemiş istisnalar oluştuğunda, ancak kod içinde yakalanmadığında, kullanıcı genel bir hata sayfasına yönlendirilir:

**Örnek 20.2**

```
<hata sayfası>
    <exception-type>UnhandledException</exception-type>
    <location>GenericError.jsp</location>
 </error-page>
```

Ayrıca inceleme sırasında HTTP 404 veya HTTP 500 hataları ile karşılaşabilirsiniz:

**Örnek 20.3**

```
<hata sayfası>
 <error-code>500</error-code>
 <location>GenericError.jsp</location>
 </error-page>
```

IIS geliştirme için 'Application_Error()' işleyicisi, uygulamanın tüm yakalanmamış istisnaları yakalamasına ve bunları tutarlı bir şekilde işlemesine olanak tanır. Bunun önemli olduğunu unutmayın, aksi takdirde istisna bilgilerinizin yanıt olarak istemciye geri gönderilme ihtimali vardır.

Apache geliştirme için, işleyici veya modüllerden dönen hatalar Apache motoru tarafından daha fazla işlem yapılmasını engelleyebilir ve sunucudan bir hata yanıtı ile sonuçlanabilir. Yanıt başlıkları, gövde, vb. işleyici/modül tarafından ayarlanabilir veya "ErrorDocument" yapılandırması kullanılarak yapılandırılabilir. Her istisnada yerelleştirilmiş bir açıklama dizesi, "Sistem Hatası - Lütfen daha sonra tekrar deneyin" gibi dostça bir hata nedeni kullanmalıyız. Kullanıcı bir hata mesajı gördüğünde, atılan istisnanın bu açıklama dizesinden türetilecek ve asla bir yığın izi, hatanın oluştuğu satır numarası, sınıf adı veya yöntem adı içerebilen istisna sınıfından türetilmeyecektir.

İstisna mesajları gibi hassas bilgileri ifşa etmeyin. Yerel dosya sistemindeki yollar gibi bilgiler ayrıcalıklı bilgiler olarak kabul edilir; tüm dahili sistem bilgileri kullanıcıdan gizlenmelidir. Daha önce de belirtildiği gibi, bir saldırgan bu bilgileri uygulamadan veya uygulamayı oluşturan bileşenlerden özel kullanıcı bilgilerini toplamak için kullanabilir.

Hata mesajlarına kişilerin isimlerini veya herhangi bir dahili iletişim bilgisini koymayın. Aşinalık düzeyine ve sosyal mühendislik istismarına yol açabilecek herhangi bir "insan" bilgisi koymayın.

### Ne İncelenmeli? Güvenle Başarısız Olmak

**Örneğin, bir uygulamanın başarısız olmasının birçok farklı nedeni olabilir:**

- İş mantığı koşullarının karşılanmamasının sonucu.
- İş mantığının bulunduğu ortamın sonucu başarısız olur.
- Uygulamanın bağlı olduğu yukarı akış veya aşağı akış sistemlerinin başarısız olması sonucu.
- Teknik donanım / fiziksel arıza.

Hatalar İspanyol Engizisyonu gibidir; popüler olarak kimse İspanyol Engizisyonunu beklemiyordu (bkz. Monty Python) ancak gerçek hayatta İspanyollar bir engizisyonun ne zaman gerçekleşeceğini biliyorlardı ve buna hazırdılar, benzer şekilde bir uygulamada, hataların gerçekleşmesini beklemeseniz de kodunuz bunların gerçekleşmesine hazır olmalıdır. Bir hata durumunda, uygulamanın "kapılarını" açık ve uygulama içindeki diğer "odaların" anahtarlarını masanın üzerinde bırakmamak önemlidir. Gereksinimlere dayalı olarak tasarlanan mantıksal bir iş akışı sırasında, bir bağlantı havuzunun kullanılamaması veya bir alt sunucunun hata vermesi gibi programatik olarak ele alınabilecek hatalar meydana gelebilir.

Bu tür başarısızlık alanları kod incelemesi sırasında incelenmelidir. Bellek, bağlantı havuzları, dosya tutamaçları gibi kaynakların serbest bırakılması gerekip gerekmediği incelenmelidir.

Kodun gözden geçirilmesi, kullanıcı oturumunun sonlandırılması veya geçersiz kılınması gereken alanların belirlenmesini de içermelidir. Bazen iş mantığı açısından ya da teknik açıdan mantıklı olmayan hatalar meydana gelebilir, örneğin oturum açmış bir kullanıcının kayıtlı olmayan bir hesaba erişmek istemesi gibi. Bu tür durumlar olası kötü niyetli faaliyetleri yansıtır. Burada, kodun herhangi bir şekilde savunmacı olup olmadığını ve kullanıcının oturum nesnesini öldürüp kullanıcıyı oturum açma sayfasına yönlendirip yönlendirmediğini gözden geçirmeliyiz. (Oturum nesnesinin her HTTP isteği üzerine incelenmesi gerektiğini unutmayın).

### Ne İncelenmeli: Potansiyel Olarak Savunmasız Kod Java

Java'da bir hata nesnesi kavramımız vardır; Exception nesnesi. Bu nesne java.lang paketinde yer alır ve Throwable nesnesinden türetilmiştir. İstisnalar, anormal bir olay meydana geldiğinde fırlatılır. Throwable'dan türetilen bir başka nesne de Error nesnesidir ve daha ciddi bir durum ortaya çıktığında fırlatılır. Error nesnesi bir catch cümlesinde yakalanabilir, ancak işlenemez, yapabileceğiniz en iyi şey Hata hakkında bazı bilgileri günlüğe kaydetmek ve ardından yeniden atmaktır.

Bilgi sızıntısı, geliştiriciler zayıf bir hata işleme stratejisi nedeniyle kullanıcı arayüzünde 'baloncuk' oluşturan bazı istisna yöntemleri kullandığında ortaya çıkabilir. **Bu yöntemler aşağıdaki gibidir**:

- printStackTrace()
- getStackTrace()

Ayrıca bilinmesi gereken önemli bir nokta da, bu yöntemlerin çıktısının, bir İstisna olduğunda System.out. println(e) ile aynı şekilde Sistem konsoluna yazdırılmasıdır. OutputStream'i JSP'nin PrintWriter nesnesine yönlendirmediğinizden emin olun, örneğin konvansiyonel olarak "out" olarak adlandırılır:

- printStackTrace(out);

java.lang.system paketini kullanarak system.err ve system.out'un nereye yazılacağını değiştirmenin mümkün olduğunu unutmayın (bash veya C/C++'da fd 1 ve 2'yi değiştirmek gibi):

- System.err alanı için setErr() ve
- System.out alanı için setOut().

Bu, hiçbir çıktının standart hataya veya standart çıkışa (istemciye geri yansıtılabilir) yazılmamasını, bunun yerine yapılandırılmış bir günlük dosyasına yazılmasını sağlamak için süreç genelinde kullanılabilir.

### C#.NET

NET'te bir System.Exception nesnesi vardır ve ApplicationException ve SystemException gibi yaygın olarak kullanılan alt nesneler kullanılır. Çalışma zamanı tarafından atılan bir SystemException nesnesini atmanız veya yakalamanız önerilmez.

Bir hata oluştuğunda, sistem veya yürütülmekte olan uygulama, Java'ya benzer şekilde hata hakkında bilgi içeren bir istisna fırlatarak bunu bildirir. Bir istisna fırlatıldıktan sonra, uygulama veya varsayılan istisna işleyicisi tarafından işlenir. Bu Exception nesnesi, Java uygulamasına benzer yöntemler içerir:

- StackTrace
- Kaynak
- Mesaj
- HelpLink

NET'te hata işleme stratejisine global hata işleme ve beklenmedik hataların ele alınması açısından bakmamız gerekir. Bu birçok şekilde yapılabilir ve bu makale kapsamlı bir liste değildir. İlk olarak, işlenmemiş bir istisna atıldığında bir Hata Olayı atılır.

**Bu, TemplateControl sınıfının bir parçasıdır, referansa bakın:**

- **http://msdn.microsoft.com/library/default.asp?url=/library/enus/cpref/html/frlrfSystemWebUITemplateControlClassErrorTopic.asp**

**Hata işleme .NET'te üç şekilde yapılabilir ve aşağıdaki sırayla yürütülür:**

- Aspx veya ilişkili codebehind sayfasında Page_Error.
- Global.asax dosyasının Application_Error bölümünde (daha önce belirtildiği gibi).
- web.config dosyasının customErrors bölümünde.

Uygulamanın hata stratejisini anlamak için bu alanlara bakılması tavsiye edilir.

**Klasik ASP**

Java ve .NET'in aksine, klasik ASP sayfalarında try-catch bloklarında yapılandırılmış hata işleme yoktur. Bunun yerine "err" adında özel bir nesneleri vardır. Bu, klasik ASP sayfalarında hata işlemeyi zorlaştırır ve hata işleyicilerde tasarım hatalarına, yarış koşullarına ve bilgi sızıntısına neden olur. Ayrıca, ASP VBScript (Visual Basic'in bir alt sürümü) kullandığından, "On Error GoTo label" gibi cümleler kullanılamaz. Klasik ASP'de hata işlemenin iki yolu vardır, birincisi "On Error Resume Next" ve "On Error GoTo 0" ile err nesnesini kullanmaktır.

**Örnek 20.4**

```
Public Function IsInteger (ByVal Number)
  Dim Res, tNumber
  Sayı = Trim(Sayı) tSayı=Sayı
  Hatada Sonraki Devam                'Bir hata oluşursa yürütmeye devam edin
  Number = CInt(Number)        'Number alfanümerik bir dizeyse Type Mismatch hatası oluşurRes =
  (err.number = 0)                    'Hata yoksa true döndürülür
  On Error GoTo 0                       'Bir hata oluşursa yürütmeyi durdurun ve hatayı
  görüntüleyin re.Pattern ="^[\+\-]? *\d+$"              'sadecebir +/- ve rakamlara izin
  verilir
  IsInteger = re.Test(tNumber) Ve Res
 End Fonksiyonu
```

İkincisi, bir hata sayfasında bir hata işleyicisi kullanmaktır (**h**ttp://support.microsoft.com/kb/299981).

**Örnek 20,5**

```
Dim ErrObj
set ErrObj = Server.GetLastError()
'Şimdi ErrObj'yi normal err nesnesi olarak kullanın
```

**C++**

C++ dilinde, herhangi bir nesne veya yerleşik tür fırlatılabilir. Bununla birlikte, kullanıcı tanımlı herhangi bir istisnanın üst tipi olarak kullanılması gereken bir STL tipi std::exception vardır, aslında bu tip STL'de ve birçok kütüphanede tüm istisnaların üst tipi olarak kullanılır. std::exception'a sahip olmak, geliştiricileri istisna kullanımına uygun bir istisna hiyerarşisi oluşturmaya teşvik eder ve tüm istisnaların bir std::exception nesnesi yakalanarak yakalanabileceği anlamına gelir ('catch (...)' bloğu yerine).

Java'dan farklı olarak, kurtaramayacağınız hatalar bile (örneğin std::bad_alloc, belleğinizin tükendiği anlamına gelir) std::exception'dan türetilir, bu nedenle bir 'catch( std::exception& e)', istisnaya erişmenize izin vermesi dışında 'catch (...)' ile benzerdir, böylece ne olduğunu öğrenebilir ve muhtemelen e.what() kullanarak bazı hata bilgilerini yazdırabilirsiniz.

C++ için birçok loglama kütüphanesi vardır, bu nedenle kod tabanınız belirli bir loglama sınıfı kullanıyorsa, hassas bilgilerin loglara yazılabileceği her yerde bu loglayıcının kullanımlarını arayın.

### Ne İncelenmeli: IIS'de Hata İşleme

Page_Error, .NET'te sunucu tarafında çalıştırılan sayfa düzeyinde bir işlemdir. Aşağıda bir örnek var ancak hata bilgisi biraz fazla bilgilendirici ve dolayısıyla kötü bir uygulama.

**Örneklem 20,6**

```
<script language="C#" runat="server">
 Sub Page_Error(Source As Object, E As EventArgs)
 Dim message As String = Request.Url.ToString()& Server.GetLastError().ToString()
 Response.Write(message) // mesajı görüntüle
 End Sub
 </script>
```

Yukarıdaki örnekteki metinde bir dizi sorun vardır. İlk olarak, HTTP isteğini kullanıcıya Request.Url.ToString() biçiminde görüntüler.

Bu noktadan önce herhangi bir veri doğrulaması yapılmadığını varsayarsak, siteler arası komut dosyası saldırılarına karşı savunmasız kalırız. İkinci olarak, hata mesajı ve yığın izi Server.GetLastError().ToString() kullanılarak kullanıcıya gösterilir ve bu da uygulamaya ilişkin dahili bilgileri ifşa eder.
Page_Error çağrıldıktan sonra Application_Error alt öğesi çağrılır.

Bir hata oluştuğunda Application_Error fonksiyonu çağrılır. Bu yöntemde hatayı günlüğe kaydedebilir ve başka bir sayfaya yönlendirebiliriz. Aslında hataları Page_Error yerine Application_Error'da yakalamak, daha önce açıklandığı gibi hataları merkezileştirmenin bir örneği olacaktır.

**Örneklem 20,7**

```
<%@ Import Namespace="System.Diagnostics" %>
 <script language="C#" runat="server">
 void Application_Error(Object sender, EventArgs e) {
  String Message = "\n\nURL: http://localhost/" + Request.Path
                  +"\n\nMESAJ:\n" + Server.GetLastError().Message
                  +"\n\nSTACK TRACE:\n" + Server.GetLastError().StackTrace;
  // Insert into Event Log
  EventLog Log = new EventLog();
  Log.Source = LogName;
  Log.WriteEntry(Message, EventLogEntryType.Error);
  Server.Redirect(Error.htm) // bu aynı zamanda hatayı
  temizleyecektir
 }
 </script>
```

Yukarıda Global.asax ve Application_Error yöntemindeki bir kod örneği yer almaktadır. Hata günlüğe kaydedilir ve ardından kullanıcı yeniden yönlendirilir. Doğrulanmamış parametreler burada Request.Path şeklinde günlüğe kaydedilir. Herhangi bir harici kaynaktan gelen doğrulanmamış girdinin günlüğe kaydedilmemesine veya görüntülenmemesine dikkat edilmelidir. "<XSS'ye bağlantı>"

Web.config, hataları işlemek için kullanılabilecek özel hata etiketlerine sahiptir. Bu en son çağrılır ve Page_error veya Application_error çağrılırsa ve işlevselliği varsa, önce bu işlevsellik yürütülür. Önceki iki işleme mekanizması yeniden yönlendirme veya temizleme yapmazsa (Response.Redirect veya Server.ClearError), bu çağrılır ve aşağıdaki gibi yapılandırılan customErrors bölümünde web.config içinde tanımlanan sayfaya yönlendirilirsiniz:

**Örneklem 20,8**

```
<customErrors mode="<On|Off|RemoteOnly>" defaultRedirect="<varsayılan yönlendirme sayfası>">
 <error statusCode="<HTTP durum kodu>" redirect="<listelenen durum kodu için özel yönlendirme sayfası>"/>
 </customErrors>
```

"mode" öznitelik değerinin "On" olması özel hataların etkinleştirildiği, "Off" olması ise özel hataların devre dışı bırakıldığı anlamına gelir. "mode" özniteliği, özel hataların yalnızca uzak istemcilere gösterileceğini ve ASP.NET hatalarının yerel ana bilgisayardan gelen isteklere gösterileceğini belirten "RemoteOnly" olarak da ayarlanabilir. "mode" özniteliği ayarlanmazsa, varsayılan olarak "RemoteOnly" değerini alır.

Bir hata oluştuğunda, yanıtın durum kodu hata öğelerinden biriyle eşleşirse, ilgili 'redirect' değeri hata sayfası olarak döndürülür. Durum kodu eşleşmezse, 'defaultRedirect' özelliğindeki hata sayfası görüntülenir. 'defaultRedirect' için herhangi bir değer ayarlanmamışsa, genel bir IIS hata sayfası döndürülür.

Bir uygulama için tamamlanan customErrors bölümünün bir örneği aşağıdaki gibidir:

**Örneklem 20,9**

```
<customErrors mode="On" defaultRedirect="error.html">
   <error statusCode="500" redirect="err500.aspx"/>
   <error statusCode="404" redirect="notHere.aspx"/>
   <error statusCode="403" redirect="notAuthz.aspx"/>
 </customErrors>
```

### Ne İncelenmeli: Apache'de Hata İşleme

**Apache'de hata mesajlarını istemciye nasıl döndüreceğiniz konusunda iki seçeneğiniz vardır:**

**1.** Hata durum kodunu req nesnesine yazabilir ve yanıtı istediğiniz şekilde görünecek şekilde yazabilir, ardından işleyicinizin 'DONE' döndürmesini sağlayabilirsiniz (bu, Apache çerçevesinin isteği işlemek için başka işleyicilere / filtrelere izin vermeyeceği ve yanıtı istemciye göndereceği anlamına gelir.

**2.** İşleyici veya filtre kodunuz, Apache çerçevesine kodlarınızın işlenmesinin sonucunu (esasen HTTP durum kodu) bildirecek önceden tanımlanmış değerleri döndürebilir. Daha sonra her hata kodu için hangi hata sayfalarının döndürüleceğini yapılandırabilirsiniz.

Tüm hata kodu işlemlerini merkezileştirmek amacıyla 2. seçenek daha mantıklı olabilir. İşleyicinizden önceden tanımlanmış belirli bir değeri döndürmek için, kullanılacak değerlerin listesi için Apache belgelerine bakın ve ardından aşağıdaki örnekte gösterildiği gibi işleyici işlevinden dönün:

**Örnek 20.10**

```
static int my_handler(request_rec *r)
{
   if ( problem_processing() )
   {
    HTTP_INTERNAL_SERVER_ERROR döndürür;
   }
    ... işleme devam etme talebi ...
}
```

Daha sonra httpd.conf dosyasında 'ErrorDocument' yönergesini kullanarak her hata kodu için hangi sayfanın döndürüleceğini belirtebilirsiniz. Bu yönergenin biçimi aşağıdaki gibidir:

- ErrorDocument <3-digit-code> <action>

... burada 3 basamaklı kod işleyici tarafından belirlenen HTTP yanıt kodudur ve eylem döndürülecek yerel veya harici bir URL veya görüntülenecek belirli bir metindir. Aşağıdaki örnekler ErrorDocument yönergeleri hakkında daha fazla bilgi ve seçenek içeren Apache ErrorDocument belgelerinden **(https://httpd.apache.org/docs/2.4/custom-error.html)** alınmıştır:

**Örnek 20.11**

```
ErrorDocument 500 "Üzgünüz, betiğimiz çöktü. Ah canım"
 ErrorDocument 500 /cgi-bin/crash-recover
 ErrorDocument 500 http://error.example.com/server_error.html
 ErrorDocument 404 /errors/not_found.html
 ErrorDocument 401 /subscription/how_to_subscribe.html
```

### Ne İncelenmeli? Hata Yönetimi için Öncü Uygulama

İstisnaları fırlatabilecek kod bir try bloğunda, istisnaları işleyen kod ise bir catch bloğunda olmalıdır. catch bloğu, catch anahtar sözcüğü ile başlayan ve ardından bir istisna türü ve gerçekleştirilecek bir eylem ile devam eden bir dizi ifadedir.

**Örnek: Java Try-Catch:**

**Örnek 20.12**

```
public class DoStuff {
   public static void Main() {
      dene {
         StreamReader sr = File.OpenText("stuff.txt");
         Console.WriteLine("Reading line {0}", sr.ReadLine());
      }
      catch(MyClassExtendedFromException e) {
         Console.WriteLine("Bir hata oluştu. Lütfen odayı terk edin");
               logerror("Hata: ", e);
      }
   }
}
```

**.NET Try-Catch**

**Örnek 20.13**

```
public void run() {
        while (!stop) {
          try {
// Çalışmayı burada gerçekleştirin
          } catch (Throwable t) {
            // İstisnayı günlüğe kaydet ve devam et
                    WriteToUser("An Error has occurred, put the kettle on");
            logger.log(Level.SEVERE, "Unexception exception", t);
```

```
        }
      }
    }
```

**C++ Try-Catch**

**Örnek 20.14**

```
void perform_fn() {
  try {
    // Çalışmayı burada gerçekleştirin

  } catch ( const MyClassExtendedFromStdException& e) {
    // İstisnayı günlüğe kaydet ve devam et
    WriteToUser("An Error has occurred, put the kettle on");
    logger.log(Level.SEVERE, "Unexception exception", e);
  }
}
```

Genel olarak, Java örneğinde temel catch(Exception) veya catch(Throwable) deyimini kullanmak yerine belirli bir istisna türünü yakalamak en iyi uygulamadır.

**Ne İncelenmeli: İstisnaları Yakalama Sırası**

Birçok dilin, atılan istisnayı bir üst sınıfla eşleştirmek anlamına gelse bile, atılan istisnayı catch cümlesiyle eşleştirmeye çalışacağını unutmayın. Ayrıca catch cümlelerinin sayfada kodlandıkları sıraya göre kontrol edildiğini unutmayın. Bu sizi belirli bir istisna türünün asla doğru şekilde ele alınamayacağı bir durumda bırakabilir, aşağıdaki örneği ele alalım; 'non_even_argument', 'std::invalid_argument'in bir alt sınıfıdır:

**Örnek 20.15**

```
class non_even_argument : public std::invalid_argument {
 public:
  explicit non_even_argument (const string& what_arg);
 };

 void do_fn()
 {
   dene
   {
     // Atılabilecek işleri gerçekleştirin
   }
   catch ( const std::invalid_argument& e )
   {
```

```
    // Genel geçersiz bağımsız değişken işleme gerçekleştirin ve başarısızlık döndürün
  }
  catch ( const non_even_argument& e )
  {
    // Argümanı eşitlemek ve işleme devam etmek için özel işlem gerçekleştirin
  }
}
```

Bu kodla ilgili sorun, bir 'non_even_argument' fırlatıldığında, 'std::invalid_argument' ile ilgilenen catch dalının 'non_even_argument' öğesinin bir üst öğesi olduğu için her zaman çalıştırılacak olması ve dolayısıyla çalışma zamanı sisteminin bunu bir eşleşme olarak değerlendirecek olmasıdır (bu aynı zamanda dilimlemeye de yol açabilir). Bu nedenle, istisna nesnelerinizin hiyerarşisinin farkında olmanız ve kodunuzda önce daha spesifik istisnalar için yakalamayı listelediğinizden emin olmanız gerekir.

Söz konusu dilin bir finally yöntemi varsa, bunu kullanın. finally yönteminin her zaman çağrılacağı garanti edilir. finally yöntemi, istisnayı atan yöntem tarafından başvurulan kaynakları serbest bırakmak için kullanılabilir. Bu çok önemlidir. Örnek olarak, bir yöntem bir bağlantı havuzundan bir veritabanı bağlantısı kazandıysa ve finally olmadan bir istisna oluştuysa, bağlantı nesnesi bir süre (zaman aşımına kadar) havuza geri döndürülmeyecektir. Bu, havuzun tükenmesine yol açabilir. finally(), herhangi bir istisna atılmasa bile çağrılır.

**Örnek 20.16**

```
void perform_fn() {
  try {
    // Çalışmayı burada gerçekleştirin

  } catch ( const MyClassExtendedFromStdException& e) {
    // İstisnayı günlüğe kaydet ve devam et
    WriteToUser("An Error has occurred, put the kettle on");
    logger.log(Level.SEVERE, "Unexception exception", e);
  }
}
```

finally() fonksiyonunun sistem kaynaklarını serbest bırakmak için kullanıldığını gösteren bir Java örneği.

### Gözden Geçirilecekler: Kaynakların serbest bırakılması ve iyi temizlik

RAII, Resource Acquisition Is Initialization'dır ve bir türün bir örneğini ilk oluşturduğunuzda, iyi bir durumda olması için tamamen (veya mümkün olduğunca) kurulması gerektiğini söylemenin bir yoludur. RAII'nin bir diğer avantajı da nesnelerin nasıl elden çıkarıldığıdır; bir nesne örneğine artık ihtiyaç duyulmadığında, nesne kapsam dışına çıktığında (C++) veya 'using' bloğu bittiğinde (Dispose yöntemini çağıran C# 'using' yönergesi veya Java 7'nin try-with-resources özelliği) kaynaklar otomatik olarak iade edilir.

RAII, programcıların (ve kütüphanelerin kullanıcılarının) nesneleri açıkça silmelerine gerek olmaması avantajına sahiptir, nesneler kendileri kaldırılır ve kendilerini kaldırma sürecinde (yıkıcı veya Dispose)

Klasik ASP sayfaları için tüm temizliğin bir fonksiyon içine alınması ve bir hata işleme içine çağrılması önerilir

"On Error Resume Next" ifadesinden sonra.

Tutarlı hata raporlaması için bir altyapı oluşturmak, hata işlemeden daha zordur. Struts, raporlanacak hata mesajları yığınını korumak için ActionMessages ve ActionErrors sınıflarını sağlar ve bu hata mesajlarını kullanıcıya görüntülemek için <html: error> gibi JSP etiketleriyle kullanılabilir.

Bir mesajın farklı bir önem derecesini farklı bir şekilde (hata, uyarı veya bilgi gibi) bildirmek için aşağıdaki görevler gereklidir:
**1.** Kaydedin, hataları uygun önem derecesi altında örneklendirin

**2.** Bu mesajları belirleyin ve tutarlı bir şekilde gösterin.

Struts ActionErrors sınıfı hata işlemeyi oldukça kolaylaştırır:

**Örnek 20.17**

```
ActionErrors errors = new ActionErrors()
 errors.add("fatal", new ActionError(" ...."));
 errors.add("hata", new ActionError(" ...."));
 errors.add("uyarı", new ActionError(" ........."));
 errors.add("bilgi", new ActionError(" ....................."));
 saveErrors(request,errors); // Bunu yapmak önemlidir
```

Artık hataları eklediğimize göre, bunları HTML sayfasındaki etiketleri kullanarak gösteriyoruz.

**Örnek 20.18**

```
<logic:messagePresent property="error">
 <html:messages property="error" id="errMsg" >
   <bean:write name="errMsg"/>
 </html:messages>
 </logic:messagePresent >
```

**Referanslar**
• Klasik ASP sayfaları için bazı IIS yapılandırmaları yapmanız gerekir, **daha fazla bilgi için http://support.microsoft.com/ kb/299981 adresini** takip edin.

• Struts'ta (web.xml) varsayılan HTTP hata sayfası işleme için bkz. https://software-security.sans.org/ blog/2010/08/11/security-misconfigurations-java-webxml-files

# GÜVENLIK UYARILARININ GÖZDEN GEÇIRILMESI

Bir şeyler ters gittiğinde kodunuz ve uygulamalarınız nasıl tepki verecek? Güvenli tasarım ve kodlama ilkelerini takip eden birçok şirket bunu saldırganların ağlarına girmesini önlemek için yapar, ancak birçok şirket, bir saldırganın b i r güvenlik açığı bulmuş olabileceği veya bir şirketin güvenlik duvarları içinde (yani İntranet içinde) kod çalıştırmak için zaten istismar ettiği senaryo için tasarım ve kodlamayı dikkate almaz.

Birçok şirket, şüpheli faaliyetleri tespit eden kalıplar için ağ ve işletim sistemi günlüklerini izlemek için SIEM günlük teknolojilerini kullanır; bu bölüm, uygulama katmanlarını ve arayüzlerini de aynı şeyi yapmaya teşvik etmeyi amaçlamaktadır.

## 21.1 Açıklama

Bu bölüm şu konulara odaklanmaktadır:

**1.** Bir sistem saldırıya uğradığında kullanıcının tepki vermesini sağlayan tasarım ve kod.
**2.** Uygulamaların ihlal edildiklerinde işaretlenmesine izin veren kavramlar.

Bir şirket güvenli tasarım ve kodlama uyguladığında, saldırganların yazılımı kötüye kullanmalarını ve erişmemeleri gereken bilgilere erişmelerini engelleme amacına sahip olacaktır. SQL enjeksiyonları, XSS, CSRF vb. için girdi doğrulama kontrolleri, saldırganların yazılıma karşı bu tür güvenlik açıklarından yararlanmasını engellemelidir. Ancak bir saldırgan savunmaları aşmaya çalıştığında veya korumalar ihlal edildiğinde yazılım nasıl tepki vermelidir?

Bir uygulamanın güvenlik sorunlarına karşı uyarı verebilmesi için neyin 'normal' olduğu ve neyin güvenlik sorunu teşkil ettiği konusunda bir bağlama ihtiyacı vardır. Bu, uygulamaya ve uygulamanın içinde çalıştığı bağlama göre farklılık gösterecektir. Genel olarak uygulamalar, aşırı günlük kaydı sistemi yavaşlatacağından, disk veya DB alanını dolduracağından ve güvenlik sorununu bulmak için tüm bilgileri filtrelemeyi çok zorlaştıracağından, meydana gelen her öğeyi günlüğe kaydetmeye çalışmamalıdır.

Aynı zamanda, yeterli bilgi izlenmez veya kaydedilmezse, mevcut bilgilere dayanarak güvenlik uyarısı yapmak çok zor olacaktır. Bu dengeyi sağlamak için bir uygulama kendi risk puanlama sistemini kullanabilir, sistem düzeyinde hangi risk tetikleyicilerinin tespit edildiğini izleyebilir (örn. geçersiz girişler, başarısız şifreler, vb.) ve farklı günlük kaydı modları kullanabilir. Normal kullanım örneğini ele alalım, bu senaryoda yalnızca kritik öğeler günlüğe kaydedilir. Ancak güvenlik riskinin arttığı algılanırsa, önemli veya güvenlik seviyesindeki öğeler günlüğe kaydedilebilir ve bunlara göre hareket edilebilir. Bu yüksek güvenlik riski, bu bölümün ilerleyen kısımlarında açıklandığı gibi daha fazla güvenlik işlevselliğini de devreye sokabilir.

Çevrimiçi bir formun (kimlik doğrulama sonrası) kullanıcının yılın bir ayını girmesine izin verdiği bir örneği ele alalım. Burada kullanıcı arayüzü, kullanıcıya ayların (Ocak'tan Aralık'a kadar) bir açılır listesini verecek şekilde tasarlanmıştır. Bu durumda, oturum açan kullanıcı 12 değerden yalnızca birini girmelidir, çünkü tipik olarak herhangi bir metin girmemelidir, bunun yerine önceden tanımlanmış açılır değerlerden birini seçmelidir.

Bu formu alan sunucu güvenli kodlama uygulamalarını takip etmişse, genellikle form alanının izin verilen 12 değerden biriyle eşleşip eşleşmediğini kontrol eder ve ardından geçerli olduğunu kabul eder. Form alanı eşleşmezse, bir hata döndürür ve sunucuda bir mesaj günlüğe kaydedebilir. Bu, saldırganın bu özel alanı istismar etmesini önler, ancak bunun bir saldırganı caydırması pek olası değildir ve diğer form alanlarına geçeceklerdir.

Bu senaryoda, kaydettiğimizden daha fazla bilgiye sahibiz. Kullanıcıya bir hata döndürdük ve belki de sunucuda bir hata kaydettik. Aslında çok daha fazlasını biliyoruz; kimliği doğrulanmış bir kullanıcı, normal kullanımda asla yapmaması gereken (açılır liste olduğu için) geçersiz bir değer girmiştir.

**Bunun birkaç nedeni olabilir:**

- Yazılımda bir hata var ve kullanıcı kötü niyetli değil.
- Bir saldırgan kullanıcıların oturum açma kimlik bilgilerini çaldı ve sisteme saldırmaya çalışıyor.
- Bir kullanıcı oturum açmıştır ancak sisteme saldırmaya çalışan bir virüs/trojan vardır.
- Bir kullanıcı oturum açmıştır ancak ortadaki adam saldırısına maruz kalmaktadır.
- Kullanıcı kötü niyetli değildir ancak bir tarayıcı eklentisi vb. ile değeri bir şekilde değiştirmiştir.

Yukarıdaki ilk durum söz konusuysa, şirket sistemlerini düzeltmek için bunu bilmelidir. Eğer 2., 3. veya 3. durum söz konusuysa, uygulama kendisini ve kullanıcıyı korumak için bazı önlemler almalıdır, örneğin kullanıcının kullanabileceği işlevselliği azaltabilir (örneğin PII görüntülenemez, şifreler değiştirilemez, finansal işlemler gerçekleştirilemez) veya güvenlik soruları veya bant dışı kimlik doğrulama gibi daha fazla kimlik doğrulamaya zorlayabilir. Sistem ayrıca kullanıcıyı beklenmedik girdinin tespit edildiği konusunda uyarabilir ve antivirüs vb. çalıştırmasını tavsiye edebilir, böylece bir saldırıyı devam ederken durdurabilir.

Açıkçası, kullanıcı işlevselliğini sınırlandırırken veya dürüst bir hata olması durumunda kullanıcıları uyarırken dikkatli olunmalıdır, bu nedenle bir risk puanı kullanmak veya oturum uyarılarını not etmek kullanılmalıdır. Örneğin, tarama oturumunda her şey normalse ve 1 karakter yerinde değilse, kullanıcının saldırıya uğradığını belirten kırmızı bir açılır kutu göstermek makul değildir, ancak bu kullanıcı için normal IP adresi değilse, alışılmadık bir zamanda giriş yapmışsa ve bu SQL enjeksiyon dizesine benzeyen 5. hatalı biçimlendirilmiş girişse, uygulamanın tepki vermesi makul olacaktır. Bu olası tepkinin yasal belgelerde belirtilmesi gerekecektir.

Başka bir senaryoda, bir saldırgan uygulama savunmalarını aşarak uygulamanın müşteri veritabanının bir kısmını ele geçirmişse, şirket bunu bilebilir mi? Veritabanındaki bilgileri ayrı tablolara bölmek verimlilik açısından mantıklıdır, ancak güvenlik açısından da, gizli bilgileri ayrı bir bölüme koymak bile saldırganın işini zorlaştırabilir. Ancak saldırgan bilgiye sahipse bunu tespit etmek zor olabilir ve uygulamalar uyarı yazılımlarına (örneğin SIEM sistemleri) yardımcı olacak adımlar atmalıdır. Birçok finans kurumu, risk puanı vermek için kullanıcının oturumunun unsurlarına bakmak için risk puanlama sistemleri kullanır; Johnny her zaman aynı IP'den bir Perşembe günü saat 6'da oturum açıyorsa, o zaman güvenilir bir modelimiz var demektir. Eğer Johnny aniden dünyanın öbür ucundaki bir IP adresinden gece 2:15'te giriş yaparsa, şifreyi 7 kez yanlış girdikten sonra, o zaman belki uzun bir yolculuktan sonra jetlag olmuştur ya da hesabı hacklenmiştir. Her iki durumda da, Johnny'nin oturum açmasına izin vermek veya bir saldırganın Johnny'nin hesabını kullanmasını engellemek için ondan bant dışı kimlik doğrulaması istemek makul olacaktır.

Uygulama bunu daha geniş bir bakış açısıyla ele alırsa, normal bir günde kullanıcıların %3'ünün farklı IP adresi, farklı saat vb. gibi daha riskli sayılabilecek bir şekilde oturum açtığını belirleyebilir. Perşembe günü bu sayının %23'e yükseldiğini görürse, kullanıcı tabanında garip bir şey mi oldu, yoksa veritabanı hacklendi mi? Bu tür bilgiler, 'daha riskli' kullanıcıların %23'ü için genel bir bant dışı kimlik doğrulaması (ve günlüklerin dahili olarak incelenmesi) uygulamak ve böylece kullanıcı için risk puanını uygulama için genel risk puanıyla birleştirmek için kullanılabilir.

Bir başka iyi seçenek de kullanıcılara asla verilmeyen kullanıcı adları ve şifreler olan 'bal hesapları'dır. Bu hesaplar tıpkı diğer kullanıcılar gibi eklenir ve DB'de saklanır, ancak aynı zamanda özel bir önbelleğe kaydedilir ve oturum açıldığında kontrol edilir. Hiçbir kullanıcıya verilmedikleri için, hiçbir kullanıcı bu hesaplarla oturum açmamalıdır, ancak bu hesaplardan biri kullanılırsa, kullanıcı adı şifre kombinasyonunun bilinmesinin tek yolu bir saldırganın veritabanını ele geçirmesidir ve bu bilgi uygulamanın daha güvenli bir duruma geçmesini ve şirketi DB'nin saldırıya uğradığı konusunda uyarmasını sağlar.

### Ne İncelenmeli

Kod modüllerini güvenlik uyarısı bakış açısıyla incelerken dikkat edilmesi gereken bazı yaygın sorunlar şunlardır:

- Uygulama saldırıya uğrayıp uğramadığını bilecek mi? Geçersiz girişleri, oturum açmaları vb. yok mu sayıyor yoksa bunları günlüğe kaydedip sisteme yönelik mevcut riskin kümülatif bir algısını yakalamak için bu durumu izliyor mu?

- Uygulama, güvenlik tehditlerine tepki vermek için günlük seviyesini otomatik olarak değiştirebilir mi? Güvenlik seviyelerini değiştirmek dinamik mi yoksa yeniden başlatma mı gerektiriyor?

- SDLC gereksinimleri veya tasarım dokümantasyonu güvenlik uyarısı oluşturacak unsurları yakalıyor mu? Bu belirleme akran değerlendirmesine tabi tutuldu mu? Test döngüsü bu senaryolar üzerinden mi yürütülüyor?

- Sistem, DB'nin elegeçirilip geçirilmediğini uygulamanın bileceği şekilde 'bal hesapları' kullanıyor mu?

- Kullanıcıların normal kullanımını kaydeden ve riskin artması durumunda belirleme veya tepki vermeye izin veren risk tabanlı bir puanlama sistemi var mı?

- Bir SIEM sistemi kullanılıyorsa, uygun tetikleyiciler belirlendi mi? Bu tetikleyici günlük mesajlarının gelecekteki geliştirmeler veya hata düzeltmeleri tarafından yanlışlıkla değiştirilmemesini sağlamak için otomatik testler oluşturuldu mu?

- Sistem, bir kullanıcının kaç başarısız oturum açma girişimi yaşadığını takip ediyor mu? Sistem buna tepki veriyor mu?

- Belirli işlevlerin (örneğin işlem başlatma, şifre değiştirme, vb.) uygulamanın şu anda içinde bulunduğu mevcut risk puanına göre farklı çalışma modları var mı?

- Güvenlik risk puanı normal seviyelere düştüğünde uygulama 'normal' çalışmaya geri dönebilir mi?

- Güvenlik riski puanı yükseldiğinde yöneticiler nasıl uyarılır? Ya da bir ihlal olduğu varsayıldığında? Operasyonel düzeyde, bu düzenli olarak test ediliyor mu? Personel değişiklikleri nasıl ele alınıyor?

# AKTIF SAVUNMA IÇIN INCELEME

Uygulama katmanında gerçekleştirilen saldırı tespiti, bir etkileşimin tüm bağlamına ve kullanıcı hakkında gelişmiş bilgilere erişebilir. Şüpheli etkinliği tespit etmek için kod içinde mantık uygulanırsa (uygulama düzeyinde bir IPS'ye benzer şekilde), uygulama neyin yüksek değerli bir sorun neyin gürültü olduğunu bilecektir. Girdi verileri zaten uygulama içinde şifresi çözülmüş ve kanonik hale getirilmiştir ve bu nedenle uygulamaya özgü izinsiz giriş tespiti gelişmiş atlatma tekniklerine karşı daha az hassastır. Bu da uygun tespit noktalarının seçilmesi koşuluyla çok düşük seviyede saldırı tespit yanlış pozitiflerine yol açar.

Temel gereklilikler dört görevi yerine getirme becerisidir:
**1.** Bir dizi şüpheli ve kötü niyetli olayın tespiti.

**2.** Saldırıları tespit etmek için bu bilginin merkezi olarak kullanılması.

**3.** Önceden tanımlanmış bir yanıtın seçilmesi.

**4.** Yanıtın yürütülmesi.

### 22.1 Açıklama

Uygulamalar, bir kullanıcının hesabındaki değişiklikler veya uygulamanın savunma duruşundaki diğer değişiklikler gibi yüksek riskli işlevleri içerebilecek bir dizi yanıt üstlenebilir. Dinamik analizde aktif savunmayı tespit etmek zor olabilir çünkü yanıtlar test uzmanı için görünmez olabilir. Kod incelemesi, bu savunmanın varlığını belirlemek için en iyi yöntemdir.

Kimlik doğrulama başarısızlık sayıları ve kilitleme ya da dosya yükleme hızı limitleri gibi diğer uygulama işlevleri 'yerelleştirilmiş' koruma mekanizmalarıdır. Bu tür bağımsız mantık, uygulama çapında bir algılama ağı ve merkezi analitik motorda bir araya getirilmedikleri sürece, bu inceleme bağlamında aktif savunma eşdeğerleri 'değildir'.
Bu bir cıvatalı araç veya kod kütüphanesi değildir, bunun yerine kuruluşların mevcut standart güvenlik kontrollerini temel alarak kendi işlerine, uygulamalarına, ortamlarına ve risk profillerine özgü kendi uygulamalarını belirlemeleri veya geliştirmeleri için bir yaklaşım sunar.

### Ne İncelenmeli

Kod incelemesinin bir savunmanın varlığını tespit etmek için kullanıldığı durumlarda, savunmanın yokluğu bir zayıflık olarak not edilmelidir. Aktif savunmanın bilinen güvenlik açıkları olan bir uygulamayı savunamayacağını ve bu nedenle bu kılavuzun diğer bölümlerinin son derece önemli olduğunu unutmayın. Kodu gözden geçiren kişi aktif savunmanın yokluğunu bir zayıflık olarak not etmelidir.

Kod incelemesinin amacı mutlaka aktif savunmanın etkinliğini belirlemek değil, sadece böyle bir yeteneğin var olup olmadığını belirlemek olabilir.

Tespit noktaları uygulamanın sunum, iş ve veri katmanlarına entegre edilebilir. Uygulamaya özel saldırı tespitinin bir saldırıyı belirleyebilmesi için tüm geçersiz kullanımı tespit etmesi gerekmez. "Sonsuz veriye" veya "büyük veriye" ihtiyaç yoktur ve bu nedenle "tespit noktalarının" konumu kaynak kod içinde çok seyrek olabilir.

Bu tür bir kodu tanımlamak için yararlı bir yaklaşım, şüpheli etkinliği tespit etmek için özel bir modülün adını bulmaktır (OWASP AppSensor gibi). Ayrıca bir şirket, Mitre'nin Ortak Saldırı Örüntüsü Numaralandırma ve Sınıflandırma (CAPEC), CAPEC-212, CAPEC-213 gibi dizelere dayalı olarak aktif savunma tespit noktalarını etiketleme politikası uygulayabilir.
OWASP AppSensor tespit noktası türü tanımlayıcıları ve CAPEC kodları genellikle yapılandırma değerlerinde (örneğin [https://code.google.com/p/appsensor/source/browse/trunk/AppSensor/src/test/resources/.esapi/ESAPI.properties?r=53 in ESAPI for Java]), parametre adlarında ve güvenlik olayı sınıflandırmasında kullanılmış olacaktır. Ayrıca, hata günlüğü ve güvenlik olayı günlüğü mekanizmalarını da inceleyin, çünkü bunlar daha sonra saldırı tespiti için kullanılabilecek verileri toplamak için kullanılıyor olabilir. Bu günlüğe kaydetme işlemini gerçekleştiren kodu veya çağrılan hizmetleri belirleyin ve kaydedilen/gönderilen olay özelliklerini inceleyin. Ardından bunların çağrıldığı tüm yerleri belirleyin.

Girdi ve çıktı doğrulamasına ilişkin hata işleme kodunun incelenmesi, büyük olasılıkla tespit noktalarının varlığını ortaya çıkaracaktır. Örneğin, beyaz liste türü bir tespit noktasında, hata işleme kod akışının yanına veya içine ek kod eklenmiş olabilir:
Bazı durumlarda saldırı tespit noktaları kara listeye alınmış girdi arar ve test başka türlü mevcut olmayabilir,

```
if ( var !Match this ) {
   Hata işleme
   Saldırı tespiti için olay kaydı
}
```

Bu yüzden yepyeni bir koda ihtiyaç vardır. Tespit noktalarının belirlenmesi, olayların kaydedildiği yerleri de ("olay deposu") bulmuş olmalıdır. Tespit noktaları bulunamazsa, aktif savunmanın varlığı hakkında fikir verebileceğinden, yanıtın yürütülmesi için kodu incelemeye devam edin.

Önceden tanımlanmış kriterlere dayalı saldırıları tespit etmek için olay deposunun gerçek zamanlı olarak veya çok sık analiz edilmesi gerekir. Kriterler yapılandırma ayarlarında tanımlanmalıdır (örneğin yapılandırma dosyalarında veya veritabanı gibi başka bir kaynaktan okunarak). Bir süreç, bir saldırının devam edip etmediğini belirlemek için olay deposunu inceleyecektir, tipik olarak bu, kimliği doğrulanmış bir kullanıcıyı tanımlamaya çalışacaktır, ancak tek bir IP adresini, IP adresleri aralığını veya bir veya daha fazla rol, belirli bir ayrıcalığa sahip kullanıcılar veya hatta tüm kullanıcılar gibi kullanıcı gruplarını da dikkate alabilir.

Bir saldırı tespit edildiğinde, yanıt önceden tanımlanmış kriterlere göre seçilecektir. Yine yapılandırma verilerinin incelenmesi, her bir tespit noktası, tespit noktası grupları veya genel eşiklerle ilgili eşikleri ortaya çıkarmalıdır.

En yaygın yanıt eylemleri kullanıcı uyarı mesajları, oturumu kapatma, hesap kilitleme ve yönetici bildirimidir. Ancak, bu yaklaşım uygulamaya bağlı olduğundan, yanıt eylemlerinin olasılıkları yalnızca uygulamanın kodlanmış yetenekleri ile sınırlıdır.

Herhangi bir global için arama kodu, yukarıda tanımlanan saldırı tanımlama/yanıtını içerir. Yanıt eylemleri (yine bir kullanıcı, IP adresi, kullanıcı grubu, vb.) genellikle uygulama tarafından başlatılır, ancak bazı durumlarda diğer uygulamalar (örneğin bir dolandırıcılık ayarını değiştirmek) veya altyapı bileşenleri (örneğin bir IP adresi aralığını engellemek) de dahil olabilir.

Yapılandırma dosyalarını ve uygulamanın gerçekleştirdiği tüm harici iletişimleri inceleyin.

**Aşağıdaki yanıt türleri kodlanmış olabilir:**
- Günlük kaydı arttı
- Yönetici bildirimi
- Diğer bildirim (örn. diğer sistem)
- Proxy
- Kullanıcı durumu değişikliği
- Kullanıcı bildirimi
- Zamanlama değişikliği
- Süreç sonlandırıldı (geleneksel savunmalarla aynı)
- İşlev devre dışı
- Hesap oturumu kapatma
- Hesap kilitleme
- Kullanıcıdan veri toplayın.

Seçilen yanıt eylemlerini sağlamak için uygulamanın ve ilgili sistem bileşenlerinin diğer yetenekleri yeniden kullanılabilir veya genişletilebilir. Bu nedenle, hesap kilitleme gibi yerelleştirilmiş güvenlik önlemleriyle ilişkili kodu gözden geçirin.

**Referanslar**
- OWASP_AppSensor_Project'te verilen uygulamalara aktif yanıt ekleme kılavuzu
- Kategori OWASP Kurumsal Güvenlik API'si

- **https://code.google.com/p/appsensor/ AppSensor tanıtım kodu**

# YARIŞ KOŞULLARI

Yarış Koşulları, bir kod parçası olması gerektiği gibi çalışmadığında ortaya çıkar (birçok güvenlik sorunu gibi). Bunlar, kodun sonlu durum makinesinin tanımlanmamış bir duruma geçmesine neden olabilecek ve aynı kaynak üzerinde birden fazla yürütme iş parçacığının çekişmesine yol açabilecek beklenmedik bir olay sıralamasının sonucudur. Birden fazla yürütme iş parçacığının bellekte veya kalıcı verilerde aynı alana etki etmesi veya bu alanı manipüle etmesi bütünlük sorunlarına yol açar.

## 23.1 Açıklama

Aynı kaynağı kullanan rakip görevlerle, kaynak adım kilidinde olmadığından veya semaforlar gibi belirteç tabanlı bir sistem kullanmadığından kolayca bir yarış koşulu elde edebiliriz. Örneğin iki süreç varsa (İş Parçacığı 1, T1) ve (İş Parçacığı 2, T2). Söz konusu kod bir X tamsayısına 10 ekler. X'in başlangıç değeri 5'tir.

**X = X + 10**

Çok iş parçacıklı bir ortamda bu kodu çevreleyen hiçbir kontrol olmadığında, kod aşağıdaki sorunla karşılaşabilir:

**T1, X'i iş parçacığı 1'deki bir yazmacın içine yerleştirir T2, X'i iş parçacığı 2'deki bir yazmacın içine yerleştirir**
**T1, T1'in kaydedicisindeki değere 10 ekler ve sonuç 15 olur T2, T2'nin kaydedicisindeki değere 10 ekler ve sonuç 15 olur T1, kaydedici değerini (15) X'e kaydeder.**
**T1, kayıt değerini (15) X içine kaydeder.**

Her iş parçacığı başlangıç değeri olan 5'e 10 eklediği için değer aslında 25 olmalıdır. Ancak T2, ekleme için X değerini almadan önce T1'in X'e kaydetmesine izin vermediği için gerçek değer 15'tir.

Bu, uygulamanın emin olmadığı bir durumda olduğu ve bu nedenle güvenliğin doğru bir şekilde uygulanamadığı tanımlanmamış davranışa yol açar.

### Ne İncelenmeli

- C#.NET'te çok iş parçacıklı ortamları kullanan kod arayın:
  - Konu
  - System.Threading
  - ThreadPool
  - System.Threading.Interlocked

- Java kodunda şunları arayın
  - java.lang.Thread
  - start()
  - stop()
  - destroy()
  - init()

- senkronize

- wait()
- notify()
- notifyAll()

• Klasik ASP için multithreading doğrudan desteklenen bir özellik değildir, bu nedenle bu tür bir yarış koşulu **yalnızca COM nesneleri kullanılırken mevcuttur.**
• Statik yöntemler ve değişkenler (nesne başına bir tane değil, sınıf başına bir tane), özellikle birden fazla iş parçacığı arasında paylaşılan bir durum varsa bir sorundur. Örneğin, Apache'de struts statik üyeleri belirli bir istekle ilgili bilgileri depolamak için kullanılmamalıdır. Bir sınıfın aynı örneği birden fazla iş parçacığı tarafından kullanılabilir ve statik üyenin değeri garanti edilemez.

• Her işlem/talep için bir tane yapıldığından, sınıf örneklerinin iş parçacığı güvenli olması gerekmez. Statik durumlar iş parçacığı güvenli olmalıdır.
- Statik değişkenlere yapılan referanslar, bunlar iş parçacığı kilitli olmalıdır.
- finally{} dışındaki yerlerde bir kilidin serbest bırakılması sorunlara neden olabilir.
- Statik durumu değiştiren statik yöntemler.

**Referanslar**
• **http://msdn2.microsoft.com/en-us/library/f857xew0(vs.71).aspx**

# TAMPON TAŞMALARI

Tampon, bilgi depolamak için ayrılmış bitişik bellek miktarıdır. Örneğin, bir programın alışveriş sepetinizin ne içerdiği veya mevcut işlemden önce hangi verilerin girildiği gibi belirli şeyleri hatırlaması gerekiyorsa. Bu bilgiler bellekte bir tampon bellekte saklanır. C, C++ gibi diller (ki
ve Objective-C son derece verimlidir, ancak kodun işlem belleğine doğrudan erişmesine (bellek tahsisi ve işaretçiler aracılığıyla) ve veri ile kontrol bilgilerinin (örneğin işlem yığınında) birbirine karışmasına izin verirler.) Bir programcı tampon bellekte hata yapar ve kullanıcı girdisinin tahsis edilen bellekten geçmesine izin verirse, kullanıcı girdisi program kontrol bilgilerinin üzerine yazabilir ve kullanıcının kodun yürütülmesini değiştirmesine izin verebilir.

Java, C#.NET, Python ve Ruby'nin, dizelerini char dizilerinde saklama biçimleri nedeniyle arabellek taşmalarına karşı savunmasız olmadıklarını unutmayın; bunların sınırları çerçeveler tarafından otomatik olarak kontrol edilir ve programcının belleğe doğrudan erişmesine izin vermezler (bunun yerine sanal makine katmanı belleği işler). Bu nedenle bu bölüm bu diller için geçerli değildir. Ancak, JNI veya 'güvenli olmayan' C# bölümleri gibi arayüzler aracılığıyla bu diller (örneğin assembly, C, C++) içinde çağrılan yerel kodun arabellek taşmalarına duyarlı olabileceğini unutmayın.

## 24.1 Açıklama

**Bir tampon ayırmak için kod belirli bir boyutta bir değişken bildirir:**

- char myBuffer[100]; // 100 char değişkenini tutacak kadar büyük

- int myIntBuf[5]; // 5 tamsayı tutacak kadar büyük

- Widget myWidgetArray[17]; // 17 Widget nesnesini tutacak kadar büyük

Otomatik sınır denetimi olmadığından, kod 23 numaralı dizi konumuna (mevcut olmayan) bir Widget eklemeye çalışabilir. Kod bunu yaptığında, derleyici 23. Widget'ın bellekte nereye yerleştirilmesi gerektiğini hesaplayacaktır (23 x sizeof(Widget) çarparak ve bunu 'myWidgetArray' işaretçisinin konumuna ekleyerek). Bu konumda bulunan başka herhangi bir nesne veya program kontrol değişkeni/kaydı üzerine yazılacaktır.

Diziler, vektörler vb. 0'dan başlayarak indekslenir, yani kaptaki ilk öğe 'myBuffer[0]' adresindedir, bu da kaptaki son öğenin dizi indeksi 100'de değil, dizi indeksi 99'da olduğu anlamına gelir. Bu, döngüler veya programlama mantığı nesnelerin belleği bozmadan son indekse yazılabileceğini varsaydığında genellikle hatalara ve 'bir eksik' hatasına yol açabilir.

**C'de ve C++ STL popüler olmadan önce, dizeler karakter dizileri olarak tutulurdu:**

- char nameString[10];

Bu, 'nameString' karakter dizisinin yukarıda açıklanan dizi indeksleme sorunlarına karşı savunmasız olduğu anlamına gelir ve dize işleme işlevlerinin çoğu (daha sonra açıklanan strcpy, strcat gibi) kullanıldığında, 10. öğenin ötesine yazma olasılığı arabellek taşmasına ve dolayısıyla bellek bozulmasına izin verir.

Örnek olarak, bir program haftanın günlerini takip etmek isteyebilir. Programcı bilgisayara 7 sayı için bir alan saklamasını söyler. Bu bir tampon örneğidir. Ancak bir ekleme girişimi olursa ne olur?

8 sayı gerçekleştirilir mi? C ve C++ gibi diller sınır denetimi yapmaz ve bu nedenle program böyle bir dilde yazılmışsa, 8. veri parçası bellekteki bir sonraki programın program alanının üzerine yazılır ve veri bozulmasına neden olur. Bu, programın en azından çökmesine neden olabilir ya da dikkatlice hazırlanmış bir taşma, taşma yükü gerçek kod olduğu için kötü amaçlı kodun çalıştırılmasına neden olabilir.

### Ne İncelenmeli? Arabellek Taşmaları

**Örnek 24.1**

```
void copyData(char *userId) {
    char smallBuffer[10]; // 10 boyutu
    strcpy (smallBuffer, userId);
 }

 int main(int argc, char *argv[]) {
   char *userId ="01234567890"; //'\n' dize sonlandırmasını eklediğinizde 12'lik ödeme yükü
                                // "01234567890" literali tarafından otomatik olarak
   eklenir copyData (userId); // bu bir tampon aşırı yüklenmesine neden olur
 }
```

strcpy (), strcat (), sprintf () ve vsprintf () gibi C kütüphanesi fonksiyonları boş sonlandırılmış dizeler üzerinde çalışır ve sınır denetimi yapmaz. gets(), sonlandırıcı bir yeni satır veya EOF (Dosya Sonu) bulunana kadar stdin'den girdiyi (bir tampona) okuyan başka bir fonksiyondur. scanf () fonksiyon ailesi de tampon taşmalarına neden olabilir.

strncpy(), strncat() ve snprintf() fonksiyonlarının kullanılması, hedef tampona kopyalanacak/vb. maksimum veri uzunluğunu belirleyen üçüncü bir 'length' parametresinin geçirilmesine izin verir. Bu, yazılmakta olan tamponun boyutuna doğru bir şekilde ayarlanırsa, hedef tamponun taşmasını önleyecektir. Ayrıca fgets() işlevinin gets() işlevinin yerine geçtiğini unutmayın. Bir tampona yazmadan önce her zaman bir dizinin sınırlarını kontrol edin. Microsoft C çalışma zamanı ayrıca '_s' son ekine sahip birçok işlevin ek sürümlerini de sağlar (strcpy_s, strcat_s, sprintf_s). Bu işlevler hata durumları için ek denetimler gerçekleştirir ve başarısızlık durumunda bir hata işleyicisi çağırır.

Aşağıdaki C kodu, kopyalama işlevi kopyalanacak karakter dizisinin uzunluğunun üçüncü argümanı olan 10'u belirten 'strncpy' tarafından gerçekleştirildiği için arabellek taşmasına karşı savunmasız değildir.

**Örnek 24.2**

```
void copyData(char *userId) {
   char smallBuffer[10]; // 10 boyutu
   strncpy(smallBuffer, userId, sizeof(smallBuffer)); // sadece ilk 10 elemanı kopyalayın
   smallBuffer[10] = 0; // Sonlandırıldığından emin olun.
 }

 int main(int argc, char *argv[]) {
   char *userId = "01234567890"; // 11 copyData'nın
   (userId) yükü;
 }
```

Günümüz C++ (C++11) programları, güvenlik açıklarını önlemeye yardımcı olan birçok STL nesnesine ve şablonuna erişime sahiptir. std::string nesnesi, çağıran kodun altta yatan işaretçilere erişmesini gerektirmez ve gerçekleştirilen işlemleri karşılamak için altta yatan dize temsilini (heap üzerindeki karakter tamponu) otomatik olarak büyütür. Bu nedenle kod, std::string nesnesi üzerinde bir arabellek taşmasına neden olamaz.

İşaretçilerle ilgili olarak (taşmalara neden olmak için başka şekillerde kullanılabilir), C++11'de yine çağıran kodun temel işaretçiyi kullanması gerekliliğini ortadan kaldıran akıllı işaretçiler vardır, bu tür işaretçiler otomatik olarak tahsis edilir ve değişken kapsam dışına çıktığında yok edilir. Bu, bellek sızıntılarını ve çift silme hatalarını önlemeye yardımcı olur. Ayrıca std::vector, std::list, vb. gibi STL kapsayıcılarının tümü belleklerini dinamik olarak ayırır, yani normal kullanım arabellek taşmalarına neden olmaz. Bu kapsayıcılara ham işaretçilerle erişmenin ya da nesneleri yeniden yorumlamanın (reinterpret_cast) hala mümkün olduğunu, bu nedenle arabellek taşmalarının mümkün olduğunu, ancak neden olmalarının daha zor olduğunu unutmayın.

Derleyiciler bellek sorunlarına da yardımcı olur, modern derleyicilerde, derlenen koda yerleştirilen ve sınır dışı bellek erişimlerini kontrol eden ince öğeler olan 'yığın kanaryaları' vardır. Bunlar kod derlenirken etkinleştirilebilir ya da otomatik olarak etkinleştirilebilir. Bu yığın kanaryalarının birçok örneği vardır ve bazı sistemler için, kuruluşların güvenlik ve performans iştahına bağlı olarak birçok yığın kanaryası seçeneği vardır. Objective-C de arabellek taşmalarına karşı hassas olduğundan Apple'ın iOS kodu için de yığın kanaryaları vardır.

Genel olarak, manuel kod incelemesi yapan bir kişinin taşma ve bire bir hata potansiyelini tespit edebileceği bariz kod örnekleri vardır, ancak diğer bellek sızıntılarını (veya sorunlarını) tespit etmek daha zor olabilir. Bu nedenle manuel kod incelemesi piyasada bulunan bellek kontrol programları ile desteklenmelidir.

### Ne İncelenmeli: Biçim İşlevi Taşmaları

Biçim işlevi, ANSI C spesifikasyonu içinde ilkel C veri türlerini insan tarafından okunabilir forma uyarlamak için kullanılabilen bir işlevdir. Neredeyse tüm C programlarında bilgi çıktısı almak, hata mesajları yazdırmak veya dizeleri işlemek için kullanılırlar.

**Tablo 23:** Biçim İşlevi Taşmaları

| Biçim Dizesi | İlgili Girdi |
|---|---|
| %x | Onaltılık değerler (unsigned int) |
| %s | Dizeler ((const) (unsigned) char*) |
| %n | Tamsayı |
| %d | Ondalık |
| %u | İşaretsiz ondalık (işaretsiz int) |

### Bazı format parametreleri:

Bu durumda %s, 'abc' parametresinin işaret ettiği değerin bir karakter dizisi olarak yazdırılmasını sağlar.

**Örneğin:**

```
char* myString ="abc";
printf ("Merhaba: %s\n",
abc);
```

Biçim dizesini biçim işlevine sağlayarak işlevin davranışını kontrol edebiliriz. Yani girdiyi bir format dizesi olarak sağlamak, uygulamamızın yapmaması gereken şeyleri yapmasını sağlar. Uygulamaya tam olarak ne yaptırabiliyoruz?

Girdi olarak %x (hex unsigned int) sağlarsak, 'printf' işlevi bu biçim dizesine ilişkin bir tamsayı bulmayı bekler, ancak herhangi bir argüman mevcut değildir. Bu durum derleme zamanında tespit edilemez. Çalışma zamanında bu sorun ortaya çıkacaktır.

Argümandaki her % için printf fonksiyonu yığında ilişkili bir değer olduğunu varsayar. Bu şekilde fonksiyon yığını aşağıya doğru yürür ve yığından ilgili değerleri okuyarak kullanıcıya yazdırır.

Biçim dizelerini kullanarak, aşağıdaki gibi bir biçim dizesi kullanarak bazı geçersiz işaretçi erişimlerini yürütebiliriz:
• printf (“%s%s%s%s%s%s%s%s%s%s%s%s”);

Daha da kötüsü 'printf()' içinde '%n' yönergesini kullanmaktır. Bu yönerge bir 'int*' alır ve o ana kadarki bayt sayısını o konuma 'yazar'.

Bu potansiyel güvenlik açığı nerede aranmalıdır. Bu sorun 'printf()' fonksiyon ailesi, "printf(), fprintf(), sprintf(), snprintf()" ile yaygındır. Ayrıca 'syslog()' (sistem günlüğü bilgilerini yazar) ve setproctitle(const char *fmt, ...); (süreç tanımlayıcı bilgilerini görüntülemek için kullanılan dizeyi ayarlar).

### Ne İncelenmeli? Tamsayı Taşmaları

Tamsayılar için veri temsili sınırlı miktarda alana sahip olacaktır, örneğin birçok dilde kısa sayı 16 bit ikiye tümleyen sayıdır, bu da maksimum 32,767 ve minimum -32,768 sayısını tutabileceği anlamına gelir. İkiye tümleyen, ilk bitin (16 bitin) sayının pozitif mi yoksa negatif mi olduğunun bir temsili olduğu anlamına gelir. Eğer ilk bit '1' ise, bu negatif bir sayıdır.

Bazı sınır sayılarının gösterimi **tablo 24**'te verilmiştir.

**Tablo 24:** Tamsayı Taşmaları

| Sayı | Temsilcilik |
|---|---|
| 32,766 | 0111111111111110 |
| 32,767 | 0111111111111111 |
| -32,768 | 1000000000000000 |
| -1 | 1111111111111111 |

32,766'ya 1 eklerseniz, yukarıda gösterilen 32,767 gösterimini veren gösterime 1 ekler. Ancak bir tane daha eklerseniz, sistem tarafından -32,768 olarak yorumlanan ilk biti (diğer bir deyişle en anlamlı biti) ayarlar.

Kısa sürede değer ekleyen veya sayan bir döngünüz (veya başka bir mantığınız) varsa, uygulama bu taşmayı yaşayabilir. Ayrıca -32,768'in altındaki değerlerin çıkarılmasının da sayının yüksek bir pozitife sarılacağı anlamına geldiğini unutmayın; buna alt taşma denir.

**Örnek 24.3**

```
#include <stdio.h>

int main(void){
        int val;
        val = 0x7fffffff;           /* 2147483647*/
        printf("val = %d (0x%x)\n", val, val);
        printf("val + 1 = %d (0x%x)\n", val + 1 , val + 1); /*İnt değerini taşır*/
        return 0;
}
```

0x7fffffff'nin ikili gösterimi 11111111111111111111111111111111 şeklindedir; bu tamsayı, işaretli bir uzun tamsayının tutabileceği en yüksek pozitif değerle başlatılır.

Burada 0x7fffffff hex değerine 1 eklediğimizde tamsayı değeri taşar ve negatif bir sayıya gider (0x7fffffff + 1 = 80000000) Ondalık olarak bu (-2147483648) olur. Bunun neden olabileceği sorunları düşünün. Derleyiciler bunu algılamayacak ve uygulama bu sorunu fark etmeyecektir.

İşaretli tamsayıları karşılaştırmalarda veya aritmetikte kullandığımızda ve ayrıca işaretli tamsayıları işaretsiz tamsayılarla karşılaştırırken bu sorunlarla karşılaşırız.

**Örnek 24,4**

```
int myArray[100];

  int fillArray(int v1, int v2){
    if(v2 > sizeof(myArray) / sizeof(int) -1 ){
      return -1; /* Çok Büyük */
    }
    myArray [v2] = v1;
    return 0;
  }
```

Burada v2 büyük bir negatif sayı ise "if" koşulu geçecektir. Bu koşul, v2'nin dizi boyutundan büyük olup olmadığını kontrol eder.

Sınır kontrolü yapılmamış olsaydı, "myArray[v2] = v1" satırı v1 değerini dizinin sınırları dışında bir konuma atayarak beklenmedik sonuçlara neden olabilirdi.

**Referanslar**

- Arabellek taşması saldırıları hakkındaki OWASP makalesine bakın.

• Arabellek taşması güvenlik açıklarının nasıl test edileceğine ilişkin OWASP Test Kılavuzu'na bakın.

• CRT'deki Güvenlik Geliştirmelerine bakın: http://msdn2.microsoft.com/en-us/library/8ef0s5kh(VS.80).aspx

JavaScript'in bilinen birçok güvenlik açığı vardır, HTML5 ve JavaScript'in günümüzde web sitelerinde daha yaygın hale gelmesi ve daha fazla web sitesinin JavaScript'e bağımlı olan duyarlı web tasarımına geçmesiyle birlikte

# MÜŞTERİ TARAFI JavaScript

kodu inceleyen kişinin hangi güvenlik açıklarını araması gerektiğini anlaması gerekir. JavaScript hızla bilgisayar korsanlarının web uygulamalarına önemli bir giriş noktası haline gelmektedir. Bu nedenle A1 Injection alt bölümüne dahil ettik.

JavaScript'teki en önemli güvenlik açıkları siteler arası komut dosyası oluşturma (XSS) ve Belge Nesne Modeli, DOM tabanlı XSS'dir.

DOM tabanlı XSS'nin tespiti zor olabilir. Bu durum aşağıdaki nedenlerden kaynaklanır.
• JavaScript genellikle fikri mülkiyeti korumak için gizlenir.
• JavaScript genellikle bant genişliği kaygısıyla sıkıştırılır.

Bu iki durumda da kod incelemesinin JavaScript'i gizlenmeden ve/veya sıkıştırılmadan önce inceleyebilmesi şiddetle tavsiye edilir. Bu, QA yazılım uzmanları ile büyük bir çekişme noktasıdır çünkü üretim durumunda olmayan kodu gözden geçiriyorsunuz.

JavaScript'in kod incelemesini zorlaştıran bir diğer husus da Microsoft .NET ve Java Server Faces gibi büyük çerçevelere dayanması ve JQuery, Knockout, Angular, Backbone gibi JavaScript çerçevelerinin kullanılmasıdır. Bu çerçeveler sorunu daha da ağırlaştırmaktadır çünkü kod yalnızca çerçevenin kaynak kodu göz önünde bulundurularak tam olarak analiz edilebilmektedir. Bu çerçeveler genellikle kod gözden geçiricinin incelemesi gereken koddan birkaç kat daha büyüktür.

Zaman ve para nedeniyle çoğu şirket bu çerçevelerin güvenli olduğunu ya da risklerin düşük ve kurum için kabul edilebilir olduğunu basitçe kabul etmektedir.

Bu zorluklar nedeniyle JavaScript için hibrit bir analiz öneriyoruz. Gerektiğinde manuel kaynaktan lavaboya doğrulama, kara kutu testi ve kusur testi ile statik analiz.

İlk olarak statik analiz kullanın. Kod İnceleme Uzmanı ve kuruluşun, olaylara bağlı davranışlar, HTML DOM ve JavaScript kodu arasındaki karmaşık bağımlılıklar ve sunucu tarafı ile eşzamansız iletişim nedeniyle statik analizin her zaman yetersiz kalacağını ve hem pozitif, hem yanlış, hem yanlış -pozitif hem de pozitif-yanlış bulgular gösterebileceğini anlaması gerekir.

Yansıyan ya da depolanan XSS'in kara kutu geleneksel yöntemlerle tespit edilmesi gerekir. Ancak bu yaklaşım DOM tabanlı XSS açıkları için işe yaramayacaktır.

Taint analizinin statik analiz motoruna dahil edilmesi gerekmektedir. Taint Analizi, kullanıcı tarafından kontrol edilebilen girdi ile 'lekelenmiş' değişkenleri belirlemeye çalışır ve bunları aynı zamanda bilinen olası savunmasız işlevlere kadar izler

bir 'lavabo' olarak. Kusurlu değişken önce sterilize edilmeden bir lavaboya aktarılırsa güvenlik açığı olarak işaretlenir. İkinci olarak, kodu gözden geçiren kişinin, tüm istemci taraflı veri doğrulamasının sunucu tarafında da doğrulandığından emin olmak için kodun JavaScript kapalıyken test edildiğinden emin olması gerekir.

**Örnek 25.1**

```
<html>
<script type="text/javascript">
var pos=document.URL.indexOf("name=")+5;
document.write( document.URL.substring(pos,document.URL.length));
</script>
<html>
```

JavaScript güvenlik açıklarına ilişkin kod örnekleri.

**Örnek 25.2**

```
var url = document.location.url;
var loginIdx = url.indexOf('login');
var loginSuffix = url.substring(loginIdx);
url ='http://mySite/html/sso/' + loginSuffix;
document.location.url = url;
```

Açıklama: Bir saldırgan kurbana **"http://hostname/welcome.html#name=<script>alert(1)</ script>"** gibi bir bağlantı göndererek kurbanın tarayıcısının enjekte edilen istemci tarafı kodunu çalıştırmasına neden olabilir. Satır 5 yanlış pozitif olabilir ve güvenli bir kod olduğu kanıtlanabilir veya taint analizi ile "Açık yönlendirme saldırısına" açık olabilir, statik analiz bu güvenlik açığının var olup olmadığını doğru bir şekilde tanımlayabilmelidir. Statik analiz yalnızca kara kutu bileşenine dayanıyorsa, bu kod güvenlik açığı olarak işaretlenecek ve kodu gözden geçiren kişinin tam bir kaynaktan lavaboya inceleme yapmasını gerektirecektir.

**Ek örnekler ve olası güvenlik riskleri Kaynak:**
document.url
**Sink:** document.write()
**Sonuçlar: Sonuçlar**:document.write("<script>malicious code</script>");

Siber suçlular aşağıdaki DOM unsurlarını kontrol edebilir
**document.url,document.location,document.referrer,window.location**

**Kaynak:** document.location
**Link:** windon.location.href
**Sonuçlar:** windon.location.href = http://www.BadGuysSite; - İstemci kodu açık yönlendirme.

**Kaynak:** document.url
**Depolama:** windows.localstorage.name

**Sink:** elem.innerHTML
**Sonuçlar:** elem.innerHTML = <value> =Stored DOM-based Cross-site Scripting

eval() güvenlik tehditlerine açıktır ve bu nedenle kullanılması önerilmez.
**Bu noktaları göz önünde bulundurun:**
**1.** Eval'a aktarılan kod, yürütücünün ayrıcalıklarıyla çalıştırılır. Bu nedenle, aktarılan kod bazı kötü niyetli kişiler tarafından etkilenebilirse, web sitenizin ayrıcalıklarıyla bir kullanıcının makinesinde kötü amaçlı kod çalıştırılmasına yol açar.
**2.** Kötü amaçlı bir kod, eval'a aktarılan kodun hangi kapsamda çağrıldığını anlayabilir.
**3.** Ayrıca JSON verilerini ayrıştırmak için eval() veya new Function() kullanmamalısınız.

Yukarıdakiler kullanılırsa güvenlik tehditlerine yol açabilir. JavaScript, kodu dinamik olarak değerlendirmek için kullanıldığında potansiyel bir güvenlik riski oluşturacaktır.

**eval('alert("Query String ' + unescape(document.location.search) + '");');**

**eval(untrusted string); Kod enjeksiyonuna veya istemci tarafı açık**

**yönlendirmeye yol açabilir. JavaScripts "new function" da potansiyel bir**

**güvenlik riski oluşturabilir.**

**JavaScript için üç geçerlilik noktası gereklidir**
**1.** Tüm mantığın sunucu tarafında olması, JavaScript doğrulamasının istemcide kapatılması
**2.** Her türlü XSS DOM Saldırısını kontrol edin, kullanıcı verilerine asla güvenmeyin, kaynağınızı ve lavabolarınızı bilin (yani, kullanıcı tarafından sağlanan girdi içeren tüm değişkenlere bakın).
**3.** Güvensiz JavaScript kütüphanelerini kontrol edin ve sık sık güncelleyin.

**Referanslar:**
- **http://docstore.mik.ua/orelly/web/jscript/ch20_04.html**
- **https://www.owasp.org/index.php/Static_Code_Analysis**
- **http://www.cs.tau.ac.il/~omertrip/fse11/paper.pdf**
- **http://www.jshint.com**


EK

# KOD İNCELEMESİNDE YAPILMASI VE YAPILMAMASI GEREKENLER

İşte bizler birer profesyoneliz. Ancak profesyoneller olarak bile kod incelemeleri yaparken kod incelemelerinin teknik yönlerinin yanı sıra kod incelemelerinin insani yönünü de dikkate aldığımızdan emin olmamız gerekir. İşte kod gözden geçirenlerin; akran geliştiricilerin dikkate alması gereken tartışma noktalarının bir listesi. Bu liste kapsamlı değildir, ancak kod incelemelerinin etkili olmasını, yıkıcı olmamasını ve bir söylem kaynağı olmasını sağlamak için bir kuruluş için bir öneri başlangıç noktasıdır. Kod incelemeleri kurum içinde b i r söylem kaynağı haline gelirse, güvenlik ve işlevsel hataları bulma etkinliği azalacak ve geliştiriciler sürecin etrafından dolaşmanın bir yolunu bulacaklardır. İyi bir kod gözden geçiricisi olmak iyi sosyal beceriler gerektirir ve tıpkı kod yazmayı öğrenmek gibi pratik gerektiren bir beceridir.

• Kod incelemesi yapmak için kodda hata bulmak zorunda değilsiniz. Eğer her zaman yorumlarınızı eleştirecek bir şey bulursanız
güvenilirliğini kaybedecektir.

• Kod incelemesini aceleye getirmeyin. Güvenlik ve işlevsellik hatalarını bulmak önemlidir, ancak diğer geliştiriciler veya ekip üyeleri
bers sizi bekliyor, bu yüzden acelecilliğinizi uygun bir acilliyetle yumuşatmaniz gerekiyor.

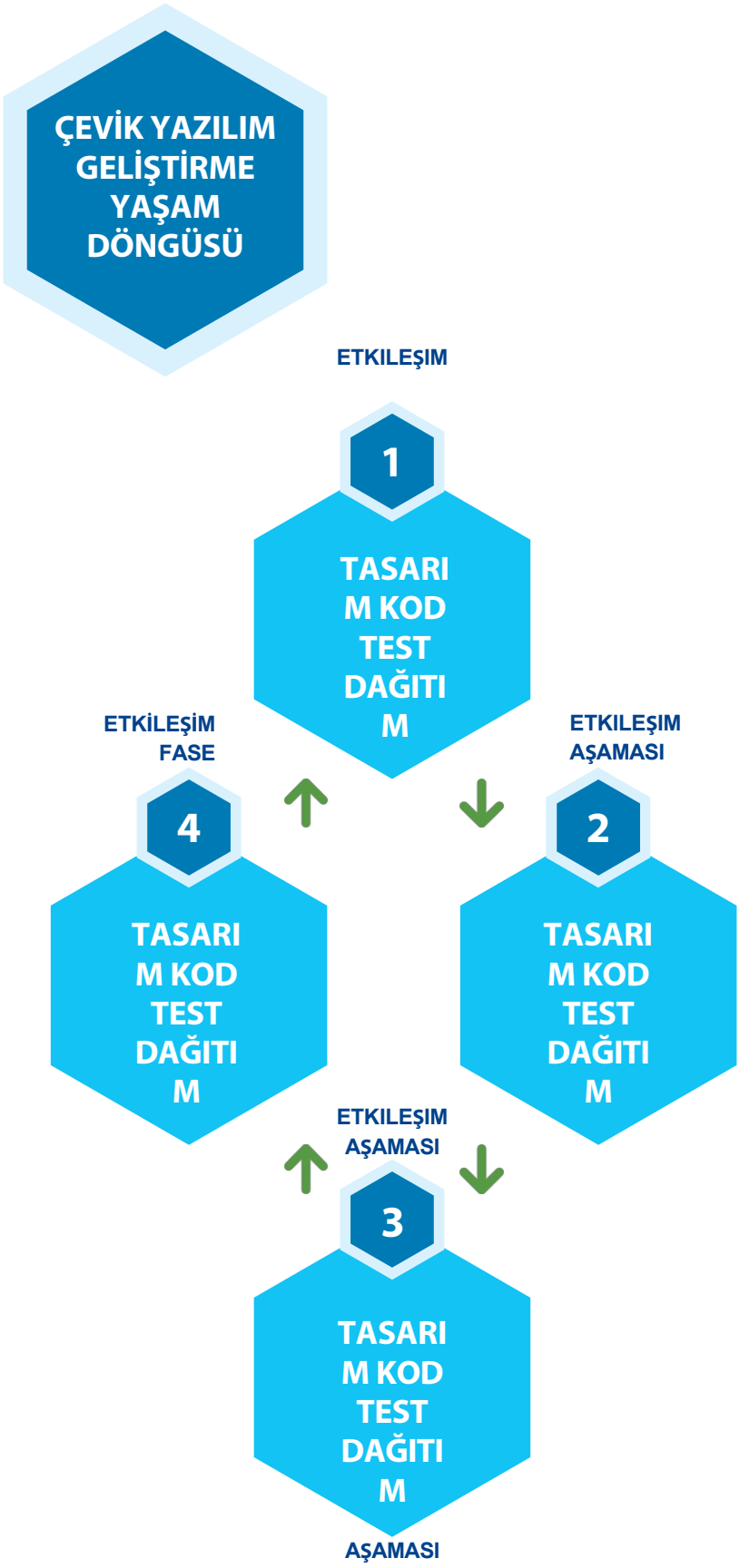
• Kodu gözden geçirirken ne beklendiğini bilmeniz gerekir. Güvenlik, işlevsellik, bakım ve onarım için mi gözden geçiriyorsunuz?
yetenek ve/veya stil? Kuruluşunuzun kod stiline ilişkin araçları ve belgeleri var mı yoksa kendi kodlama stilinizi mi kullanıyorsunuz? Kuruluşunuz geliştiricilere kuruluşun kendi kodlama standartlarına göre kabul edilemez kodlama standartlarını işaretlemeleri için araçlar veriyor mu?

• Bir kod incelemesine başlamadan önce kuruluşunuzun ortaya çıkabilecek anlaşmazlıkları çözmek için tanımlanmış bir yolu var mı?
kod incelemesinde geliştirici ve kod gözden geçiren tarafından nasıl değerlendirilir?

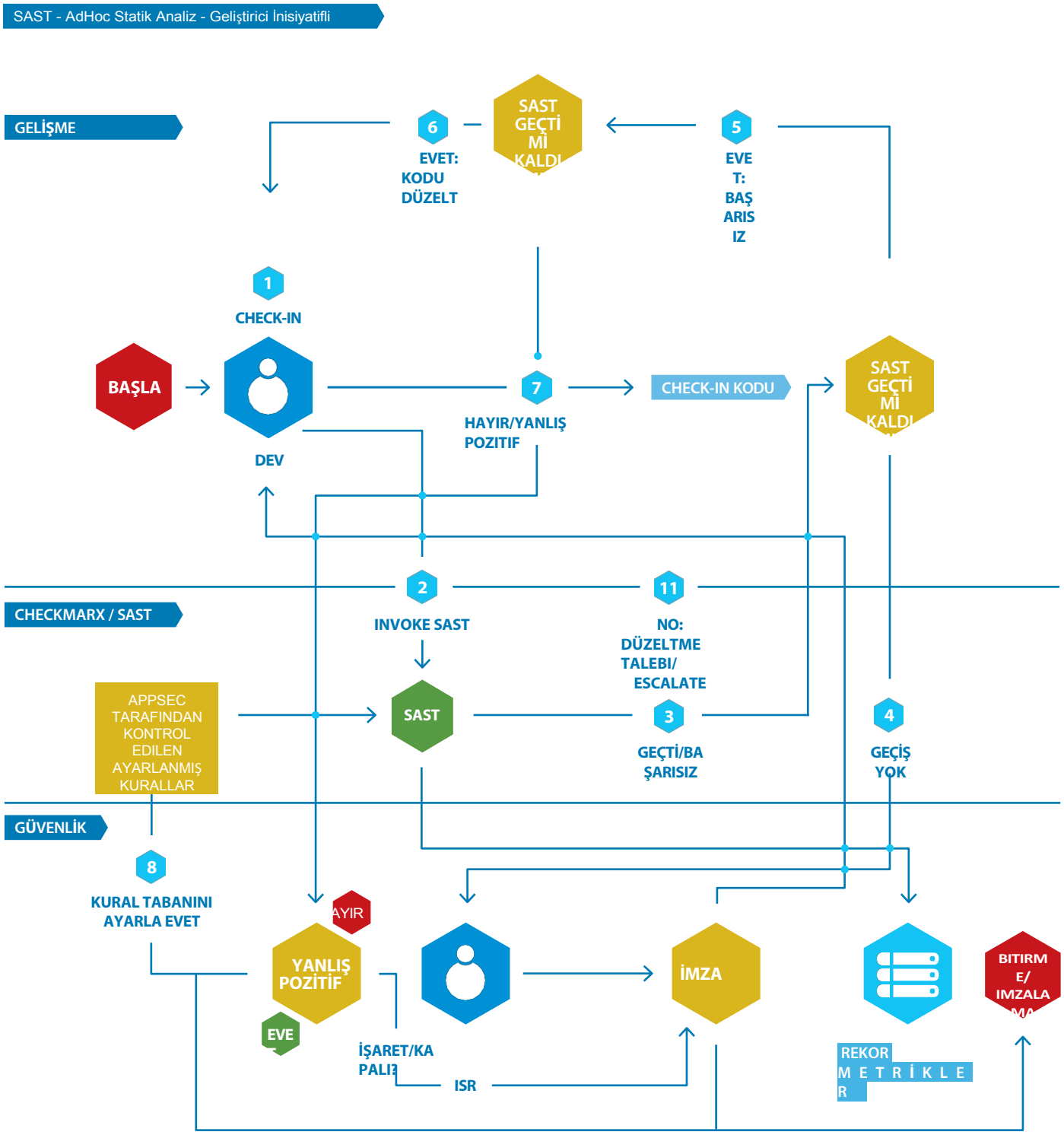
• Kod gözden geçiricinin kod gözden geçirme sonucunda üretmesi gereken tanımlanmış bir eser seti var mı?

• Kod incelemesi sırasında kodun değiştirilmesi gerektiğinde kod inceleme süreci nedir?

• Kod gözden geçiren kişi gözden geçirilen kodun alan bilgisi hakkında bilgili mi? Kod gözden geçiren kişinin kodun ana faaliyet alanı hakkında bilgi sahibi olması durumunda kod gözden geçirmelerinin en etkili şekilde yapıldığına dair çok sayıda kanıt bulunmaktadır, örneğin endüstri ve devlet için uyumluluk düzenlemeleri, iş işlevselliği, riskler vb. **Çevik Yazılım Geliştirme Yaşam Döngüsü**


ÇEVİK YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ
ETKILEŞIM
1
TASARIM KOD TEST DAĞITIM
ETKİLEŞİM FASE
ETKILEŞIM AŞAMASI
4
TASARIM KOD TEST DAĞITIM
2
TASARIM KOD TEST DAĞITIM
ETKILEŞIM AŞAMASI
3
TASARIM KOD TEST DAĞITIM
AŞAMASI

Güvenliği çevik sdlc süreç akışına entegre etmek zordur. Kuruluşun güvenlik ekibinin sürekli katılımına ve her ekipte iyi eğitimli kodlayıcılarla güvenliğe adanmışlığa ihtiyacı olacaktır.




## Sürekli Entegrasyon ve Test Güdümlü Geliştirme

'Sürekli Entegrasyon' terimi Extreme Programming geliştirme süreci ile ortaya çıkmıştır. Bugün SDLC Agile'ın en iyi uygulamalarından biridir. CI, geliştiricilerin kaynak kontrol yönetimine kodu kontrol etmesini gerektirir

uygulamasını (scm) günde birkaç kez kontrol eder. Otomatik bir derleme sunucusu ve uygulaması her girişi doğrular. Bunun avantajı, ekip üyelerinin yazılım geliştirme sürecinin başlarında derleme sorunlarını hızlı bir şekilde tespit edebilmesidir.
CI'ın Kod Gözden Geçiricisi için dezavantajı, kod düzgün bir şekilde oluşturulabilirken; yazılım güvenlik açıkları hala mevcut olabilir. Kod incelemesi check-in sürecinin bir parçası olabilir, ancak inceleme yalnızca kodun kuruluşun minimum standartlarını karşıladığından emin olmak için yapılabilir. Kod incelemesi, kod incelemesi yapmak için ek zaman harcanması gereken konulara ilişkin risk değerlendirme yaklaşımına sahip güvenli bir kod incelemesi değildir.

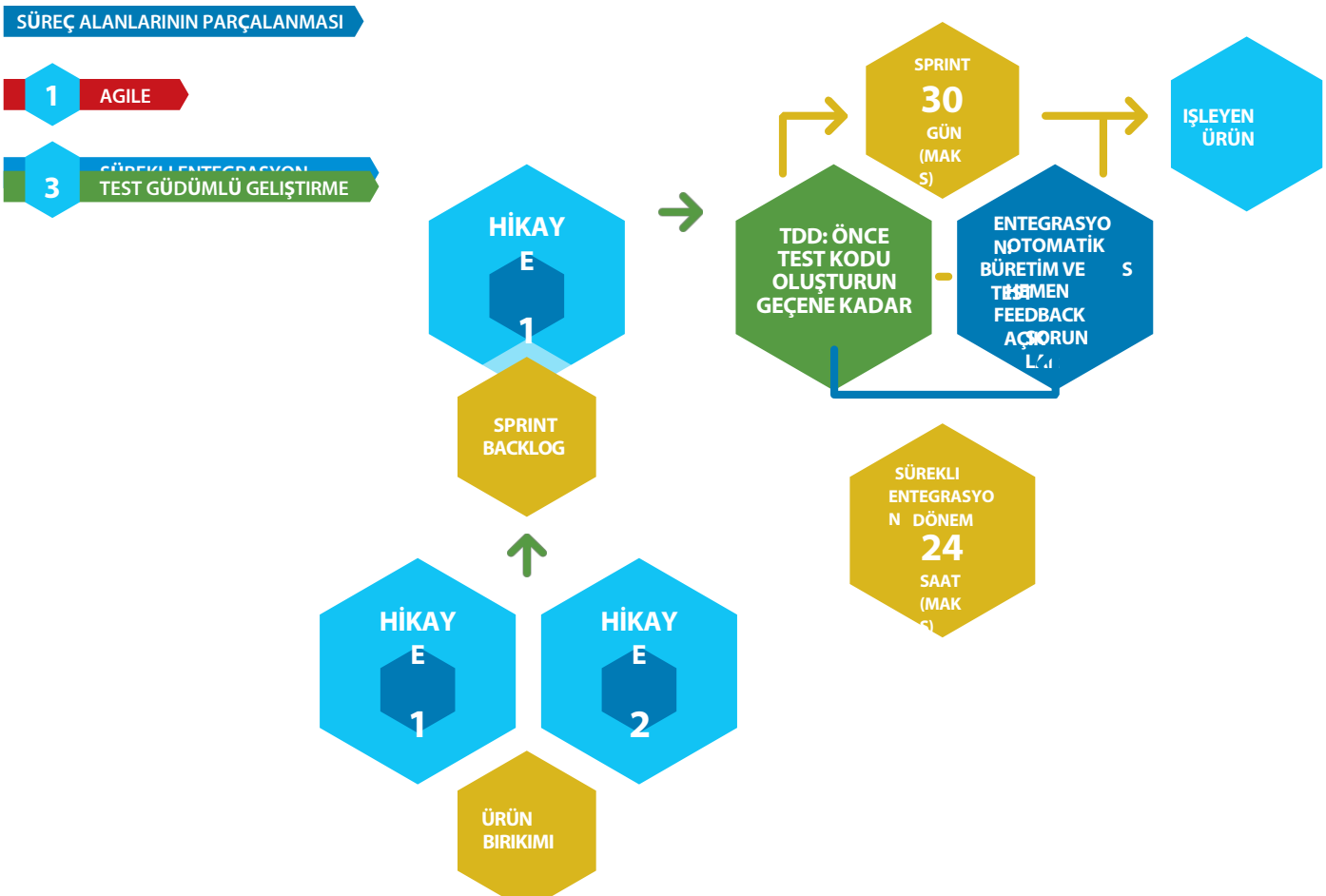
Kod incelemesi için ikinci dezavantaj, kuruluşun Çevik süreçle hızlı hareket etmesi nedeniyle bir güvenlik açığına izin veren bir tasarım hatası ortaya çıkabilir ve güvenlik açıkları dağıtılabilir.

**Kodu gözden geçiren kişi için kırmızı bayrak...**

**1.** Hiçbir kullanıcı hikayesi riske dayalı güvenlik açıklarından bahsetmez.
**2.** Kullanıcı hikayeleri kaynak ve lavaboları açıkça tanımlamaz.
**3.** Uygulama için herhangi bir risk değerlendirmesi yapılmamıştır.

### Süreç Alanlarının Ayrıştırılması

CI gibi 'Test Güdümlü Geliştirme' terimi de Extreme Programming geliştirme süreci ile ortaya çıkmıştır. Günümüzde CI gibi SDLC Agile'ın en iyi uygulamalarından biridir. TDD, geliştiricilerin çok kısa bir geliştirme döngüsünün tekrarına güvenmelerini gerektirir. İlk adım, geliştiricinin ihtiyaç duyulan bir iyileştirmeyi veya yeni bir işlevselliği tanımlayan otomatik bir test senaryosu yazmasıdır. TDD'nin bu ilk adımı başlangıçta testi başarısız kılar. Sonraki adımlarda geliştiriciler testi geçmek için minimum miktarda kod oluşturur. Son olarak geliştirici yeni kodu kuruluşun kabul edilebilir kodlama standardına göre yeniden düzenler.



# KOD INCELEME KONTROL LISTESI

| KATEGORİ | AÇIKLAMA | GEÇTİ | BAŞARISIZ |
|---|---|---|---|
| Genel | Arka kapı/açığa çıkan iş mantığı sınıfları var mı? | | |
| İş Mantığı ve Tasarım | İş mantığı ile ilgili kullanılmayan konfigürasyonlar var mı? | | |
| İş Mantığı ve Tasarım | İş mantığı yöntemlerini tanımlamak için istek parametreleri kullanılıyorsa, kullanıcı ayrıcalıkları ve bunlara izin verilen yöntemler/eylemler arasında uygun bir eşleme var mı? | | |
| İş Mantığı ve Tasarım | Kullanıcı girdilerine bağlanan form nesnelerinde açıkta olmayan örnek değişkenlerin bulunup bulunmadığını kontrol edin. Varsa, varsayılan değerlere sahip olup olmadıklarını kontrol edin. | | |
| İş Mantığı ve Tasarım | Kullanıcı girdilerine bağlanan form nesnelerinde açıkta olmayan örnek değişkenler olup olmadığını kontrol edin. Varsa, form bağlamadan önce başlatılıp başlatılmadıklarını kontrol edin. | | |
| Yetkilendirme | Kimlik doğrulama ve yetkilendirme kontrolünün yerleşimi doğru mu? | | |
| Yetkilendirme | Geçersiz istekten sonra yürütme durduruldu mu/sonlandırıldı mı? Yani kimlik doğrulama/otomatikleştirme kontrolü başarısız olduğunda? | | |
| Yetkilendirme | Kontroller doğru şekilde uygulanıyor mu? Herhangi bir arka kapı parametresi var mı? | | |
| Yetkilendirme | Kontrol, web kök dizini içindeki tüm gerekli dosya ve klasörlere uygulandı mı? | | |
| Yetkilendirme | Girdileri işlemeden önce güvenlik kontrolleri yapılıyor mu? | | |
| İş Mantığı ve Tasarım | Kullanıcı girdilerine bağlanan form nesnelerinde açıkta olmayan örnek değişkenlerin bulunup bulunmadığını kontrol edin. Varsa, varsayılan değerlere sahip olup olmadıklarını kontrol edin. | | |
| İş Mantığı ve Tasarım | Kullanıcı girdilerine bağlanan form nesnelerinde açıkta olmayan örnek değişkenler olup olmadığını kontrol edin. Varsa, form bağlamadan önce başlatılıp başlatılmadıklarını kontrol edin. | | |
| Yetkilendirme | Geçersiz istekten sonra yürütme durduruldu mu/sonlandırıldı mı? Yani kimlik doğrulama/otomatikleştirme kontrolü başarısız olduğunda? | | |
| İş Mantığı ve Tasarım | Kontroller doğru uygulanıyor mu? Herhangi bir arka kapı parametresi var mı? | | |
| İş Mantığı ve Tasarım | Kontrol, web kök dizini içindeki tüm gerekli dosya ve klasörlere uygulandı mı? | | |
| İş Mantığı ve Tasarım | Access- ALL gibi herhangi bir varsayılan yapılandırma var mı? | | |
| İş Mantığı ve Tasarım | Yapılandırma tüm dosyalara ve kullanıcılara uygulanıyor mu? | | |
| Yetkilendirme | Konteyner tarafından yönetilen kimlik doğrulama durumunda - Kimlik doğrulama yalnızca web yöntemlerine mi dayanıyor? | | |
| Yetkilendirme | Konteyner tarafından yönetilen kimlik doğrulama durumunda - Kimlik doğrulama tüm kaynaklara uygulanıyor mu? | | |
| Oturum Yönetimi | Tasarım oturumları güvenli bir şekilde ele alıyor mu? | | |
| Yetkilendirme | Konteyner tarafından yönetilen kimlik doğrulama durumunda - Kimlik doğrulama yalnızca web yöntemlerine mi dayanıyor? | | |
| Yetkilendirme | Parola Karmaşıklık Kontrolü parola üzerinde zorunlu mu? | | |
| Kriptografi | Şifre şifrelenmiş bir biçimde mi saklanıyor? | | |
| Yetkilendirme | Parola kullanıcıya açıklanıyor mu / bir dosyaya / günlüklere / konsola yazılıyor mu? | | |

| KATEGORİ | AÇIKLAMA | GEÇTİ | BAŞARISIZ |
|---|---|---|---|
| Kriptografi | Veritabanı kimlik bilgileri şifrelenmiş bir biçimde mi saklanıyor? | | |
| İş Mantığı ve Tasarım | Tasarım düz dosyalar gibi zayıf veri depolarını destekliyor mu? | | |
| İş Mantığı ve Tasarım | Merkezi doğrulama tüm taleplere ve tüm girdilere uygulanıyor mu? | | |
| İş Mantığı ve Tasarım | Merkezi doğrulama kontrolü tüm özel karakterleri engelliyor mu? | | |
| İş Mantığı ve Tasarım | Doğrulamadan atlanan özel bir talep türü var mı? | | |
| İş Mantığı ve Tasarım | Tasarım, parametreler veya özelliklerin doğrulanması için herhangi bir dışlama listesi tutuyor mu? | | |
| Imput Doğrulama | Tüm güvenilmeyen girdiler doğrulanmış mı?<br>Girdi verileri tür, uzunluk, biçim ve aralık açısından kısıtlanır ve doğrulanır. | | |
| Kriptografi | Veriler şifreli kanal üzerinden mi gönderiliyor? Uygulama harici bağlantılar yapmak için HTTPClient kullanıyor mu? | | |
| Oturum Yönetimi | Tasarım, bileşenler/modüller arasında oturum paylaşımını içeriyor mu? Oturum her iki uçta da doğru şekilde doğrulanmış mı? | | |
| İş Mantığı ve Tasarım | Tasarım harici bağlantılar/komutlar için herhangi bir yükseltilmiş işletim sistemi/sistem ayrıcalığı kullanıyor mu? | | |
| İş Mantığı ve Tasarım | Kullanılan API'lerde/Teknolojide bilinen herhangi bir kusur(lar) var mı? Örneğin: DWR | | |
| İş Mantığı ve Tasarım | Tasarım çerçevesi herhangi bir dahili güvenlik kontrolü sağlıyor mu? ASP NET MVC'deki <%: ASP.NET MVC'deki %> gibi mi? Uygulama bu kontrollerden yararlanıyor mu? | | |
| İş Mantığı ve Tasarım | Ayrıcalıklar mümkün olduğunca azaltılıyor mu? | | |
| İş Mantığı ve Tasarım | Program zarif bir şekilde başarısız olacak şekilde tasarlandı mı? | | |
| Günlüğe Kaydetme ve Denetim | Günlükler kişisel bilgileri, parolaları veya diğer hassas bilgileri kaydediyor mu? | | |
| Günlüğe Kaydetme ve Denetim | Denetim günlükleri bağlantı denemelerini (hem başarılı hem de başarısız) kaydediyor mu? | | |
| Günlüğe Kaydetme ve Denetim | İstenmeyen/kötü niyetli davranışlara yönelik denetim günlüklerini okumak için bir süreç(ler) var mı? | | |
| Kriptografi | Tüm PI ve hassas bilgiler ağ üzerinden şifreli bir şekilde mi gönderiliyor? | | |
| Yetkilendirme | Uygulama tasarımı sunucu kimlik doğrulamasını gerektiriyor mu (sahteciliğe karşı önlem)? | | |
| Yetkilendirme | Uygulama parola süresinin dolmasını destekliyor mu? | | |
| Kriptografi | Uygulama, karma veya kriptografik için özel şemalar kullanıyor mu? | | |

| KATEGORİ | AÇIKLAMA | GEÇTİ | BAŞARISIZ |
|---|---|---|---|
| Kriptografi | Uygulama tarafından kullanılan kriptografik işlevler bu protokollerin en son sürümüne sahip mi, yamalı mı ve güncel tutulmaları için bir süreç mevcut mu? | | |
| Genel | Uygulama işlevleri tarafından kullanılan harici kütüphaneler, araçlar, eklentiler bu protokollerin en son sürümüne sahip mi, yamalı mı ve bunları güncel tutmak için bir süreç var mı? | | |
| Genel | Güvenlik sırları (parolalar gibi) içeren sınıflara yalnızca korumalı API'ler aracılığıyla erişilebilir | | |
| Kriptografi | Doğrulamadan atlanan özel bir talep türü var mı? | | |
| Genel | Güvenlik sırları (parolalar gibi) içeren sınıflara yalnızca korumalı API'ler aracılığıyla erişilebilir | | |
| Kriptografi | Anahtarlar şifreli olarak tutulmaz. | | |
| Genel | Düz metin sırları uzun süreler boyunca bellekte saklanmaz. | | |
| Genel | Dizi sınırları kontrol edilir. | | |
| Kullanıcı Yönetimi ve Kimlik Doğrulama | Kullanıcı ve rol bazlı ayrıcalıklar belgelenmiştir | | |
| Genel | Uygulama tarafından kullanılan tüm hassas bilgiler tanımlandı | | |
| Kullanıcı Yönetimi ve Kimlik Doğrulama | Kimlik doğrulama çerezleri kalıcı değildir | | |
| Kullanıcı Yönetimi ve Kimlik Doğrulama | Kimlik doğrulama çerezleri şifrelenir | | |
| Kullanıcı Yönetimi ve Kimlik Doğrulama | Kimlik doğrulama bilgileri HTTP GET tarafından aktarılmaz | | |
| Kullanıcı Yönetimi ve Kimlik Doğrulama | Yetkilendirme kontrolleri ayrıntılıdır (sayfa ve dizin düzeyinde) | | |
| Kullanıcı Yönetimi ve Kimlik Doğrulama | Açıkça tanımlanmış rollere dayalı yetkilendirme | | |
| Kullanıcı Yönetimi ve Kimlik Doğrulama | Yetkilendirme düzgün çalışır ve parametre manipülasyonu ile aşılamaz | | |
| Kullanıcı Yönetimi ve Kimlik Doğrulama | Yetkilendirme, çerez manipülasyonu ile atlanamaz | | |
| Oturum Yönetimi | URL'lerde oturum parametreleri geçirilmez | | |
| Oturum Yönetimi | Oturum çerezleri makul bir süre içinde sona erer | | |
| Oturum Yönetimi | Oturum çerezleri şifrelenir | | |
| Oturum Yönetimi | Oturum verileri doğrulanır | | |
| Oturum Yönetimi | Oturum kimliği karmaşık | | |
| Oturum Yönetimi | Oturum depolama güvenlidir | | |

| KATEGORİ | AÇIKLAMA | GEÇTİ | BAŞARISIZ |
|---|---|---|---|
| Oturum Yönetimi | Oturum hareketsizlik zaman aşımları uygulanır | | |
| Veri Yönetimi | Veriler sunucu tarafında doğrulanır | | |
| Veri Yönetimi | HTTP üstbilgileri her istek için doğrulanır | | |
| İş Mantığı ve Tasarım | Tüm giriş noktaları ve güven sınırları tasarım tarafından belirlendi mi ve risk analizi raporunda yer alıyor mu? | | |
| Veri Yönetimi | Tüm XML giriş verileri üzerinde anlaşmaya varılmış bir şemaya göre doğrulanıyor mu? | | |
| Veri Yönetimi | Girdi olarak sağlanan güvenilmeyen verileri içeren çıktı doğru kodlama türüne sahip mi (URL kodlaması, HTML kodlaması)? | | |
| Veri Yönetimi | Uygulama tarafından çıktısı alınan tüm verilere doğru kodlama uygulandı mı? | | |
| Web Hizmetleri | Uygulama WSDL'nin dinamik olarak oluşturulmasına ihtiyaç duymuyorsa Web hizmeti dokümantasyon protokolüne sahiptir protokolü devre dışı bırakılır. | | |
| Web Hizmetleri | Web Hizmetleri Tanımlama Dili'ndeki (WSDL) web hizmeti uç noktalarının adresinin geçerliliği kontrol edilir | | |
| Web Hizmetleri | Gereksiz olan web hizmeti protokolleri devre dışı bırakılır (HTTP GET ve HTTP POST | | |

# TEHDİT MODELLEME ÖRNEĞİ

### Adım 1 Uygulamayı Ayrıştırın

Uygulamayı ayrıştırmanın amacı, uygulamayı ve dış varlıklarla nasıl etkileşime girdiğini anlamaktır. Bilgi toplama ve dokümantasyon bu amaca ulaşmayı sağlar. Bilgi toplama süreci, doğru bilgilerin toplanmasını sağlayan açıkça tanımlanmış bir yapı kullanılarak gerçekleştirilir. Bu yapı aynı zamanda tehdit modelini oluşturmak için bilgilerin nasıl belgelenmesi gerektiğini de tanımlar.

**Şekil 13**

**Genel Bilgiler**

Tehdit modelindeki ilk madde tehdit modeline ilişkin genel bilgilerdir. Bu bilgiler aşağıdakileri içermelidir:

**1. Uygulama Adı:** Uygulamanın adı

**2. Uygulama Sürümü:** Uygulamanın sürümü

**3. Açıklama:** Uygulamanın üst düzey bir açıklaması

**4. Belge Sahibi:** Tehdit modelleme belgesinin sahibi

**5. Katılımcılar:** Bu uygulama için tehdit modelleme sürecine dahil olan katılımcılar

**6. Gözden geçiren:** Tehdit modelini gözden geçiren(ler)

**Şekil 14**


### Açıklama

Üniversite kütüphanesi web sitesi, kütüphanecilere ve kütüphane kullanıcılarına (öğrenciler ve üniversite personeli) çevrimiçi hizmetler sağlamak için bir web sitesinin ilk uygulamasıdır. Bu web sitesinin ilk uygulaması olduğu için işlevselliği sınırlı olacaktır. Uygulamanın üç kullanıcısı olacaktır:

**1.** Öğrenciler

**2.** Personel

**3.** Kütüphaneciler Personel ve öğrenciler giriş yapıp kitap arayabilecek ve personel üyeleri kitap talebinde bulunabilecektir. Kütüphaneciler giriş yapabilecek, kitap ekleyebilecek, kullanıcı ekleyebilecek ve kitap arayabilecektir.

**Şekil 15**

**Giriş Noktaları**

**Giriş noktaları aşağıdaki şekilde belgelenmelidir:**

**1. KIMLIK**
Giriş noktasına atanan benzersiz bir kimlik. Bu, giriş noktasını tespit edilen tehditler veya güvenlik açıkları ile çapraz referanslamak için kullanılacaktır. Katman giriş noktaları söz konusu olduğunda, büyük, küçük notasyonu kullanılmalıdır.

**2. İsim**
Giriş noktasını ve amacını tanımlayan açıklayıcı bir ad.

**3. Açıklama**
Giriş noktasında gerçekleşen etkileşimi veya işlemi detaylandıran metinsel bir açıklama.

**4. Güven Seviyeleri**
Giriş noktasında gerekli olan erişim seviyesi burada belgelenmiştir. Bunlar, belgenin ilerleyen bölümlerinde tanımlanan güven seviyeleri ile çapraz referanslandırılacaktır.

**Şekil 16**

**Varlıklar**

**Varlıklar tehdit modelinde aşağıdaki şekilde belgelenmiştir:**

**1. KIMLIK**
Her bir varlığı tanımlamak için benzersiz bir kimlik atanır. Bu, varlığı tanımlanan herhangi bir tehdit veya güvenlik açığı ile çapraz referanslamak için kullanılacaktır.

**2. İsim**
Varlığı açıkça tanımlayan açıklayıcı bir ad.

**3. Açıklama**
Varlığın ne olduğu ve neden korunması gerektiğine dair metinsel bir açıklama.

**4. Güven Seviyeleri**
Giriş noktasına erişmek için gereken erişim seviyesi burada belgelenir. Bunlar bir sonraki adımda tanımlanan güven seviyeleri ile çapraz referanslandırılacaktır.

**Şekil 17**

## Güven Seviyeleri

Güven seviyeleri tehdit modelinde aşağıdaki şekilde belgelenmiştir:

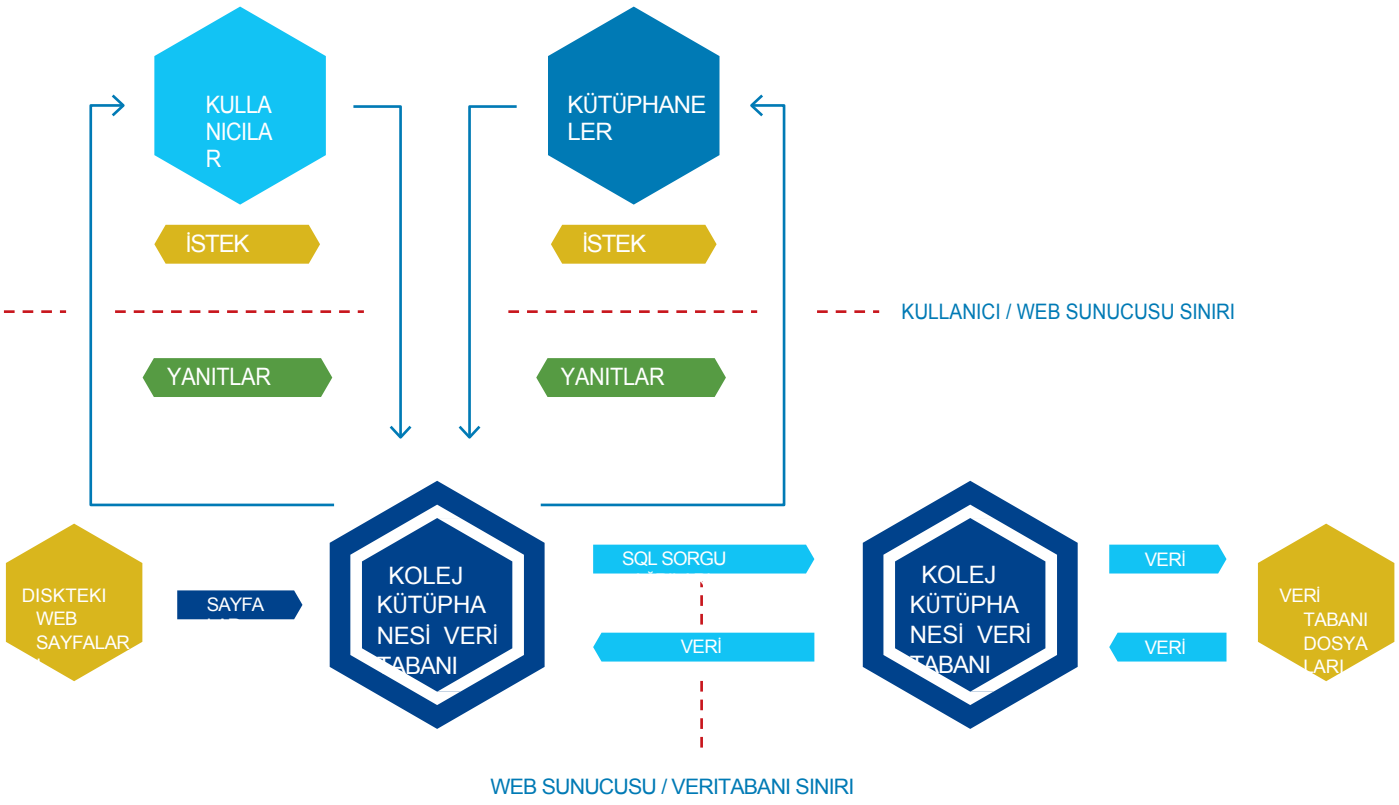
**1. KIMLIK**
Her güven seviyesine benzersiz bir numara atanır. Bu, güven seviyesini giriş noktaları ve varlıklarla çapraz referanslamak için kullanılır.

**2. İsim**
Bu güven düzeyine sahip harici varlıkların tanımlanmasına olanak tanıyan açıklayıcı bir ad.

**3. Açıklama**
Güven seviyesinin metinsel açıklaması, güven seviyesinin verildiği harici kuruluşu detaylandırır.

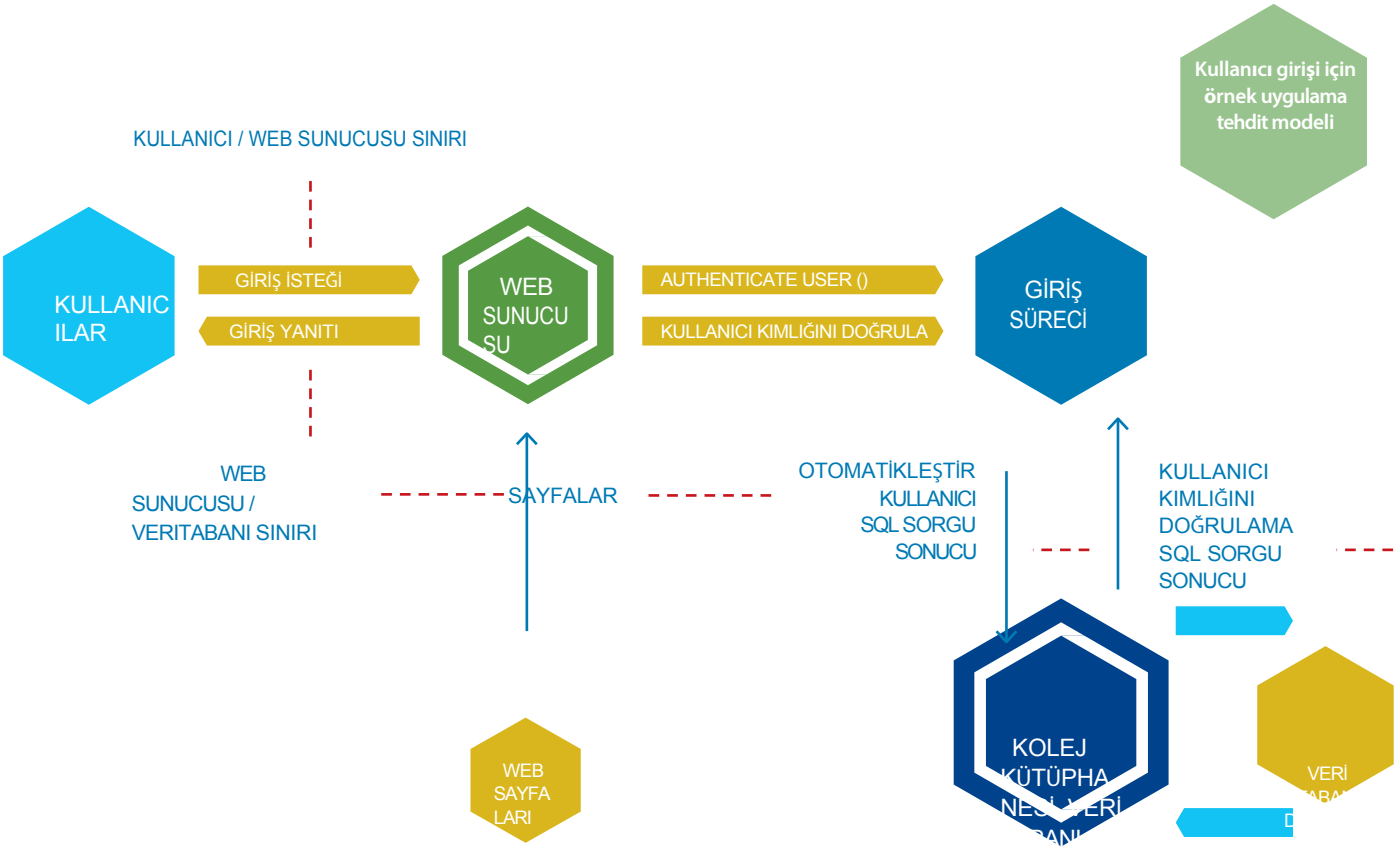
Üniversite kütüphanesi web sitesi mimarisi ve tasarımında öğrenilen anlayış kullanılarak, veri akış diyagramı şekil X'de gösterildiği gibi oluşturulabilir.

**Şekil 18**



Özellikle kullanıcı girişi veri akış diyagramı **şekil 19**'daki gibi görünecektir.

**Şekil 19**




### Tehdit Modelleme Örneği: Adım 2a Tehdit Kategorizasyonu

Tehditlerin belirlenmesindeki ilk adım bir tehdit kategorizasyonunun benimsenmesidir. Bir tehdit kategorizasyonu, tehditlerin uygulamada sistematik olarak yapılandırılmış ve tekrarlanabilir bir şekilde tanımlanabilmesi için karşılık gelen örneklerle birlikte bir dizi tehdit kategorisi sağlar.

#### Stride

Bu kategorilerde düzenlenen genel tehditlerin bir tehdit listesi, örnekler ve etkilenen güvenlik kontrolleri ile birlikte aşağıdaki tabloda verilmiştir:

## Tehdit Modelleme Örneği: Adım 2b Tehditlerin Sıralanması

Tehditler risk faktörleri perspektifinden sıralanma eğilimindedir. Belirlenen çeşitli tehditlerin oluşturduğu risk faktörünü belirleyerek, hangi tehditlerin öncelikle azaltılması gerektiğine karar vermek gibi bir risk azaltma stratejisini desteklemek için önceliklendirilmiş bir tehdit listesi oluşturmak mümkündür. Hangi tehditlerin Yüksek, Orta veya Düşük risk olarak sıralanabileceğini belirlemek için farklı risk faktörleri kullanılabilir. Genel olarak, tehdit risk modelleri riskleri modellemek için farklı faktörler kullanır.

### Microsoft DREAD tehdit-risk sıralama modeli

Üniversite kütüphanesi web sitesine başvurarak, aşağıdaki gibi kullanım durumlarıyla ilgili örnek tehditleri belgelemek mümkündür:

**Tehdit: Kötü niyetli kullanıcılar öğrencilerin, öğretim üyelerinin ve kütüphanecilerin gizli bilgilerini görüntüler.**

**1. Hasar potansiyeli**
İtibarın yanı sıra mali ve yasal sorumluluğa yönelik tehdit:8

**2. Tekrar Üretilebilirlik**
Tamamen yeniden üretilebilir:10

3. İstismar Edilebilirlik
Aynı alt ağ üzerinde olmayı veya bir yönlendiriciyi tehlikeye atmış olmayı gerektirir:7

**4. Etkilenen kullanıcılar**
Tüm kullanıcıları etkiler:10

**5. Keşfedilebilirlik**
Kolayca öğrenilebilir:10
Genel DREAD puanı: (8+10+7+10+10) / 5 = 9
Bu durumda 10 puan üzerinden 9 puan almak kesinlikle yüksek riskli bir tehdittir.

# CODE CRAWLING

Bu ekte, aşağıdaki programlama dillerinde kod taramasının nasıl yapılacağına dair pratik örnekler verilmektedir:

- .Net
- Java
- ASP
- C++/Apache

### NET'te Kod Arama

Öncelikle kişinin metin araması yapmak için kullanabileceği araçlara aşina olması, ardından ne arayacağını bilmesi gerekir.

"Kullanıcı", "Şifre", "Pswd", "Anahtar", "Http", vb. gibi yaygın kalıpları veya anahtar kelimeleri arayarak kodu tarayabilirsiniz. Bu, VS'deki "Dosyalarda Bul" aracı kullanılarak veya findstring aşağıdaki gibi kullanılarak gerçekleştirilebilir: findstr /s /m /i /d:c:\projects\code- base\sec "http" *.*

### HTTP İstek Dizeleri

Harici kaynaklardan gelen istekler, güvenlik kodu incelemesinin önemli bir alanıdır. Alınan tüm HTTP isteklerinin kompozisyon, maksimum ve minimum uzunluk için veri doğrulamasının yapıldığından ve verilerin parametre beyaz listesinin kapsamına girip girmediğinden emin olmamız gerekir. Sonuç olarak bu, güvenliğin etkinleştirildiğinden emin olmak için bakılması gereken önemli bir alandır.

| ARANACAK DIZE | | | |
|---|---|---|---|
| request.accesstypes | request.httpmethod | request.cookies | request.url |
| request.browser | request.querystring | request.certificate | request.urlreferrer |
| request.files | request.item | request.rawurl | request.useragent |
| request.headers | istek.form | request.servervariables | request.userlanguages |
| request.TotalBytes | request.BinaryRead | | |


### HTML Çıktısı

Burada müşteriye verilen yanıtları arıyoruz. Doğrulanmayan veya veri doğrulaması olmadan harici girdiyi yankılayan yanıtlar incelenmesi gereken kilit alanlardır. Birçok istemci tarafı saldırısı zayıf yanıt doğrulamasından kaynaklanır.

| ARANACAK DIZE | | | |
|---|---|---|---|
| yanıt.yaz | HttpUtility | HtmlEncode | UrlEncode |
| innerText | innerHTML | <%= | |

## SQL & Veritabanı

Bir veritabanının kodun neresinde yer alabileceğini bulmak, kod incelemesinin önemli bir yönüdür. Veritabanı koduna bakmak, uygulamanın SQL enjeksiyonuna karşı savunmasız olup olmadığını belirlemeye yardımcı olacaktır. Bunun bir yönü, kodun SqlParameter, OleDbParameter veya OdbcParameter(System.Data.SqlClient) kullandığını doğrulamaktır. Bunlar yazılır ve parametreleri veritabanındaki çalıştırılabilir kod olarak değil, gerçek değer olarak ele alır.

| ARANACAK DIZE | | | |
|---|---|---|---|
| exec sp_ | şuradan seçin | Ekle | güncelleme |
| delete from where | silme | sp_'yi çalıştır | exec xp_ |
| exec @ | execute @ | yürütme beyanı | executeSQL |
| setfilter | executeQuery | GetQueryResultInXML | adodb |
| sqloledb | sql sunucusu | sürücü | Sunucu.CreateObject |
| .Provider | System.Data.sql | ADODB.recordset | Yeni OleDbConnection |
| ExecuteReader | Veri Kaynağı | SqlCommand | Microsoft.Jet |
| SqlDataReader | ExecuteReader | SqlDataAdapter | Saklı Prosedür |

## Çerezler

Çerez manipülasyonu, oturum kaçırma/sabitleme ve parametre manipülasyonu gibi çeşitli uygulama güvenliği istismarlarının anahtarı olabilir. Oturum güvenliğini etkileyebileceğinden, çerez işlevselliği ile ilgili tüm kodlar incelenmelidir.

| ARANACAK DIZE | | |
|---|---|---|
| System.Net.Cookie | Sadece HTTPOnly | document.cookie |


## HTML Etiketleri

Aşağıdaki HTML etiketlerinin birçoğu çapraz site komut dosyası oluşturma gibi istemci tarafı saldırıları için kullanılabilir. Bu etiketlerin kullanıldığı bağlamın incelenmesi ve bu etiketlerin bir web uygulaması içinde görüntülenmesi ve kullanılmasıyla ilgili veri doğrulamasının incelenmesi önemlidir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| HtmlEncode | URLEncode | <applet> | <frameset> |
| <embed> | <frame> | <html> | <iframe> |
| <img> | <style> | <katman> | <ilayer> |
| <meta> | <nesne> | <çerçeve güvenliği | <iframe güvenliği |

## Giriş Kontrolleri

Aşağıdaki giriş kontrolleri, web uygulaması form alanlarını üretmek ve görüntülemek için kullanılan sunucu sınıflarıdır. Bu tür referansların aranması, uygulamaya giriş noktalarının bulunmasına yardımcı olur.

| ARANACAK DIZE | | | |
|---|---|---|---|
| htmlcontrols.htmlinputhidden | webcontrols.hiddenfield | webcontrols.hyperlink | webcontrols.textbox |
| webcontrols.label | webcontrols.linkbutton | webcontrols.listbox | webcontrols.checkboxlist |
| webcontrols.dropdownlist | | | |

## WEB.config

NET Framework, yapılandırma ayarlarını tanımlamak için .config dosyalarını kullanır. .config dosyaları metin tabanlı XML dosyalarıdır. Tek bir sistemde birçok .config dosyası bulunabilir ve genellikle de bulunur. Web uygulamaları, uygulamanın kök dizininde bulunan bir web.config dosyasına başvurur. ASP.NET uygulamaları için web.config, uygulamanın çalışmasının çoğu yönü hakkında bilgi içerir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| requestEncoding | responseEncoding | İz | yetkilendirme |
| derleme | webcontrols.linkbutton | webcontrols.listbox | webcontrols.checkboxlist |
| webcontrols.dropdownlist | CustomErrors | httpCookies | httpHandlers |
| httpRuntime | sessionState | maxRequestLength | Hata Ayıklama |
| form koruma | appSettings | ConfigurationSettings | appSettings |
| connectionStrings | kimlik doğrulama modu | İzin ver | Reddet |
| Kimlik Bilgileri | kimlik taklidi | zaman aşımı | uzak |

## global.asax

Gerekirse her uygulamanın kendi global.asax dosyası vardır. Global.asax, komut dosyaları kullanan bir uygulama için olay kodunu ve değerlerini ayarlar. Uygulama değişkenlerinin hassas bilgiler içermediğinden emin olunmalıdır, çünkü bunlar tüm uygulama ve içindeki tüm kullanıcılar tarafından erişilebilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| Application_OnAuthenticateRequest | Application_OnAuthorizeRequest | Session_OnStart | Session_OnEnd |

### Günlük kaydı

Günlük kaydı bir bilgi sızıntısı kaynağı olabilir. Günlük alt sistemine yapılan tüm çağrıları incelemek ve herhangi bir hassas bilginin günlüğe kaydedilip kaydedilmediğini belirlemek önemlidir. Yaygın hatalar, kimlik doğrulama işlevindeki parolalarla birlikte kullanıcı kimliğinin günlüğe kaydedilmesi veya hassas veriler içerebilecek veritabanı isteklerinin günlüğe kaydedilmesidir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| log4net | Console.WriteLine | System.Diagnostics.Debug | System.Diagnostics.Trace |

### machine.config

machine.config dosyasındaki birçok değişkenin belirli bir uygulama için web.config dosyasında geçersiz kılınabilmesi önemlidir.

| ARANACAK DIZE | | |
|---|---|---|
| validateRequest | enableViewState | enableViewStateMac |

### İş Parçacıkları ve Eşzamanlılık

Eşzamanlılık sorunları yarış koşullarına neden olabileceğinden çok iş parçacıklı işlevler içeren kodun bulunması güvenlik açıklarına neden olabilir. Thread anahtar sözcüğü yeni thread nesnelerinin oluşturulduğu yerdir. Hassas güvenlik bilgilerini tutan statik global değişkenler kullanan kodlar oturum sorunlarına neden olabilir. Statik kurucular kullanan kodlar da iş parçacıkları arasında sorunlara neden olabilir. Dispose yönteminin senkronize edilmemesi, birkaç iş parçacığının aynı anda Dispose'u çağırması durumunda sorunlara neden olabilir, bu da kaynak serbest bırakma sorunlarına neden olabilir.

| ARANACAK DIZE | |
|---|---|
| Konu | İmha edin |

### Sınıf Tasarımı

Public ve Sealed sınıf seviyesindeki tasarımla ilgilidir. Türetilmesi amaçlanmayan sınıflar mühürlenmelidir. Tüm sınıf alanlarının bir nedenle Public olduğundan emin olun. Gerekli olmayan hiçbir şeyi açığa çıkarmayın.

| ARANACAK DIZE | |
|---|---|
| Kamu | Mühürlü |


## Yansıma, Serileştirme

Kod çalışma zamanında dinamik olarak oluşturulabilir. Harici girdinin bir fonksiyonu olarak dinamik şekilde oluşturulan kod sorunlara yol açabilir. Kod hassas veriler içeriyorsa serileştirilmesi gerekir mi?

| ARANACAK DIZE | | | |
|---|---|---|---|
| Serializable | AllowPartiallyTrustedCallersAttribute | GetObjectData | System.Reflection |
| StrongNameIdentity | StrongNameIdentityPermission | | |

## İstisnalar ve Hatalar

Bir istisna durumunda catch bloklarının kullanıcıya bilgi sızdırmadığından emin olun. Kaynaklarla uğraşırken finally bloğunun kullanıldığından emin olun. İzlemenin etkin olması bilgi sızıntısı açısından iyi değildir. Özelleştirilmiş hataların düzgün bir şekilde uygulandığından emin olun

| ARANACAK DIZE | | | |
|---|---|---|---|
| yakalamak | Sonunda | izleme etkin | customErrors modu |

## Kriptografi

Kriptografi kullanılıyorsa, AES veya 3DES gibi yeterince güçlü bir şifre kullanılıyor mu? Hangi boyutta anahtar kullanılıyor? Ne kadar büyükse o kadar iyidir. Hash işlemi nerede gerçekleştiriliyor? Kalıcı hale getirilen parolalar hashleniyor mu? Öyle olmalıdır. Rastgele sayılar nasıl üretilir? PRNG "yeterince rastgele" mi?

| ARANACAK DIZE | | | |
|---|---|---|---|
| RNGCryptoServiceProvider | SHA | MD5 | base64 |
| DES | RC2 | System.Random | Rastgele |
| xor | Sistem.Güvenlik.Kriptografi | | |

## Depolama

Hassas verileri bellekte saklıyorsanız, aşağıdakileri kullanmamanız önerilir.

| ARANACAK DIZE | |
|---|---|
| SecureString | ProtectedMemory |

### Yetkilendirme, Onaylama ve Geri Alma

.Net kod erişim güvenlik iznini atlamak mı? İyi bir fikir değil. Aşağıda, CLR dışında yönetilmeyen kodu çağırmak gibi potansiyel olarak tehlikeli izinlerin bir listesi bulunmaktadır.

| ARANACAK DIZE | | | |
|---|---|---|---|
| RequestMinimum | RequestOptional | Assert | Debug.Assert |
| CodeAccessPermission | ÜyeErişimi | ControlAppDomain | UnmanagedCode |
| SkipVerification | ControlEvidence | SerializationFormatter | ControlPrincipal |
| ControlDomainPolicy | ControlPolicy | | |

### Eski Yöntemler

Her bağlamda kontrol edilmesi gereken bazı standart işlevler aşağıdakileri içerir.

| ARANACAK DIZE | |
|---|---|
| printf | strcpy |

### Java'da Kod Arama

### Giriş ve Çıkış Akışları

Bunlar bir uygulamaya veri okumak için kullanılır. Bir uygulamaya potansiyel giriş noktaları olabilirler. Giriş noktaları harici bir kaynaktan olabilir ve araştırılmalıdır. Bunlar ayrıca yol geçiş saldırılarında veya DoS saldırılarında da kullanılabilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| FileInputStream | ObjectInputStream | FilterInputStream | PipedInputStream |
| SequenceInputStream | StringBufferInputStream | BufferedReader | ByteArrayInputStream |
| java.io.FileOutputStream | Dosya | ObjectInputStream | PipedInputStream |
| StreamTokenizer | getResourceAsStream | java.io.FileReader | java.io.FileWriter |
| java.io.RandomAccessFile | java.io.Dosya | renameTo | Mkdir |

## Servletler

Bu API çağrıları parametre/başlık/URL/çerez kurcalama, HTTP Yanıt Bölme ve bilgi sızıntısı için bir yol olabilir. Bu tür API'lerin birçoğu parametreleri doğrudan HTTP isteklerinden aldığı için yakından incelenmelidirler.

| ARANACAK DIZE | | | |
|---|---|---|---|
| javax.servlet.* | getParameterNames | getParameterValues | getParameter |
| getParameterMap | getScheme | getProtocol | getContentType |
| getServerName | getUzakAdres | getRemoteHost | getRealPath |
| getLocalName | getAttribute | getAttributeNames | getLocalAddr |
| getAuthType | getUzakKullanıcı | getCookies | isSecure |
| HttpServletRequest | getQueryString | getHeaderNames | getHeaders |
| getPrincipal | getUserPrincipal | isUserInRole | getInputStream |
| getOutputStream | getWriter | addCookie | addHeader |
| setHeader | setAttribute | putValue | javax.servlet.http.Çerez |
| getName | getPath | getDomain | getYorum |
| getMethod | getPath | getReader | getRealPath |
| getRequestURI | getRequestURL | getServerName | getValue |
| getValueNames | getRequestedSessionId | | |

## Çapraz Site Komut Dosyası

Bu API çağrıları, Çapraz Site Komut Dosyası güvenlik açıklarının kaynağı olabileceğinden kod incelemesinde kontrol edilmelidir.

| ARANACAK DIZE | |
|---|---|
| javax.servlet.ServletOutputStream.print | strcpy |

## Yanıt Bölme

Yanıt bölme, bir saldırganın başlıklara fazladan CRLF ekleyerek yanıt gövdesinin kontrolünü ele geçirmesine olanak tanır. HTTP'de başlıklar ve gövdeler 2 CRLF karakteri ile ayrılır ve bu nedenle bir saldırganın girdisi bir yanıt başlığında kullanılırsa ve bu girdi 2 CRLF içeriyorsa, CRLF'lerden sonraki her şey yanıt gövdesi olarak yorumlanır. Kod incelemesinde, işlevselliğin başlıklara konan tüm bilgileri sterilize ettiğinden emin olun.

| ARANACAK DIZE | | |
|---|---|---|
| javax.servlet.http.HttpServletResponse.sendRedirect | strcpy | setHeader |

## Yeniden Yönlendirme

Bir uygulama bir yönlendirme yanıtı gönderdiğinde, ilgili mantığın bir saldırgan girdisi tarafından manipüle edilemeyeceğinden emin olun. Özellikle de yönlendirmenin nereye gideceğini belirlemek için girdi kullanıldığında.

| ARANACAK DIZE | | | |
|---|---|---|---|
| sendRedirect | setStatus | addHeader | etHeader |

## SQL & Veritabanı

Java veritabanı ile ilgili kodun aranması, incelenmekte olan uygulamanın kalıcılık katmanında yer alan sınıfların/yöntemlerin belirlenmesine yardımcı olacaktır.

| ARANACAK DIZE | | | |
|---|---|---|---|
| java.sql.Connection.prepareStatement | java.sql.ResultSet.getObject | seçin | Ekle |
| java.sql.Statement.executeUpdate | java.sql.Statement.addBatch | yürütmek | yürütme beyanı |
| createStatement | java.sql.ResultSet.getString | executeQuery | jdbc |
| java.sql.Statement.executeQuery | java.sql.Statement.execute | silme | güncelleme |
| java.sql.Connection.prepareCall | | | |

## SSL

Noktadan noktaya şifreleme için bir araç olarak SSL kullanan kod aranıyor. Aşağıdaki parçalar SSL işlevselliğinin nerede geliştirildiğini göstermelidir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| com.sun.net.ssl | SSLContext | SSLSocketFactory | TrustManagerFactory |
| HttpsURLConnection | KeyManagerFactory | | |

## Oturum Yönetimi

Aşağıdaki API'ler oturum yönetimini kontrol ettiklerinde kod incelemesinde kontrol edilmelidir.

| ARANACAK DIZE | | |
|---|---|---|
| getSession | geçersiz | getId |

## Eski Etkileşim

Burada komut enjeksiyon saldırılarına veya işletim sistemi enjeksiyon saldırılarına karşı savunmasız olabiliriz. Java'nın yerel işletim sistemine bağlanması ciddi sorunlara neden olabilir ve potansiyel olarak sunucunun tamamen ele geçirilmesine yol açabilir.

| ARANACAK DIZE | | |
|---|---|---|
| java.lang.Runtime.exec | java.lang.Runtime.getRuntime | getId |

## Günlük kaydı

Kişinin uygulamasında yer alan aşağıdaki kodu inceleyerek bazı bilgi sızıntılarına rastlayabiliriz.

| ARANACAK DIZE | | | |
|---|---|---|---|
| java.io.PrintStream.write | log4j | jLo | Oduncu |
| MonoLog | qflog | just4log | log4Ant |
| JDLabAgent | | | |


## Ajax ve JavaScript

Ajax kullanımına ve olası JavaScript sorunlarına bakın:

| ARANACAK DIZE | | | |
|---|---|---|---|
| document.write | eval | document.cookie | pencere.konum |
| belge.URL | belge.URL | | |

## Klasik ASP'de Kod Arama

ASP'deki Giriş API'leri genellikle istekten girdiyi almak için kullanılır, bu nedenle kod incelemesi bu isteklerin (ve bağımlı mantığın) bir saldırgan tarafından manipüle edilemeyeceğinden emin olmalıdır. Çıktı API'leri ASP tarafından son kullanıcıya gönderilecek yanıt gövdesini yazmak için kullanılır, bu nedenle kod incelemesi bu isteklerin uygun bir şekilde kullanıldığını ve hiçbir hassas bilginin döndürülemeyeceğini kontrol etmelidir. Çerezler de bilgi sızıntısı kaynağı olabilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| İstek | Request.QueryString | Talep.Form | Request.ServerVariables |
| Query_String | gizli | şunları içerir | .inc |
| Yanıt.Yaz | Response.BinaryWrite | <%= | .cookies |

## Hata İşleme

Bir uygulamadaki hataların düzgün bir şekilde işlendiğinden emin olun, aksi takdirde bir saldırgan uygulamayı manipüle etmek için hata koşullarını kullanabilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| err. | Server.GetLastError | Hatada Sonraki Devam | Hatada GoTo 0 |

## URL'deki bilgiler

Bu API'ler, istekteki URL nesnesinden bilgi çıkarmak için kullanılır. Kod incelemesi, URL'den çıkarılan bilgilerin sterilize edildiğini kontrol etmelidir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| location.href | location.replace | method="GET" | Hatada GoTo 0 |

## Veritabanı

Bu API'ler, SQL saldırılarına yol açabilecek bir veritabanı ile etkileşim kurmak için kullanılabilir. Kod incelemesi, bu API çağrılarının sterilize edilmiş girdi kullandığını kontrol edebilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| commandText | şuradan seçin | güncelleme | içine yerleştir |
| delete from where | IRowSet | yürütmek | .execute |
| .open | ADODB. | Komut türü | ICommand |

## Oturum

Bu API çağrıları ASP uygulamaları içindeki oturumu kontrol edebilir.

| ARANACAK DIZE | | |
|---|---|---|
| oturum.zaman aşımı | oturum.terk | session.removeall |

## DOS Önleme ve Günlüğe Kaydetme

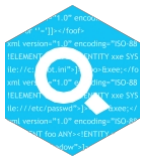
Aşağıdaki ASP API'leri uygulamaya karşı DOS saldırılarını önlemeye yardımcı olabilir. Bir günlüğe bilgi sızdırmak bir saldırganın işine yarayabilir, bu nedenle aşağıdaki API çağrısı, günlüklere hassas bilgi yazılmadığından emin olmak için kod incelemesinde kontrol edilebilir.

| ARANACAK DIZE | | |
|---|---|---|
| server.ScriptTimeout | IsClientConnected | WriteEntry |

## Yeniden Yönlendirme

Reddetme işleminin ne zaman ve nerede gerçekleşeceğini kontrol etmek için saldırgan girdisine izin vermeyin.

| ARANACAK DIZE | | | |
|---|---|---|---|
| Response.AddHeader | Response.AppendHeader | Yanıt.Yönlendirme | Yanıt.Durum |
| Response.StatusCode | Sunucu.Transfer | Sunucu.Çalıştır | |



## Javascript ve AJAX'ta Kod Arama

Ajax ve JavaScript, işlevselliği istemci tarafına geri getirmiş, bu da bir dizi eski güvenlik sorununu yeniden ön plana çıkarmıştır. Aşağıdaki anahtar kelimeler kullanıcı durumunu manipüle etmek veya tarayıcıyı kontrol etmek için kullanılan API çağrılarıyla ilgilidir. AJAX ve diğer Web 2.0 paradigmalarının ortaya çıkışı, güvenlik endişelerini istemci tarafına geri itti, ancak geleneksel sunucu tarafı güvenlik endişelerini hariç tutmadı. **Ajax kullanımına ve olası JavaScript sorunlarına bakın.**

| ARANACAK DIZE | | | |
|---|---|---|---|
| eval | document.cookie | document.referrer | document.attachEvent |
| document.body | document.body.innerHtml | document.body.innerText | document.close |
| document.create | document.execCommand | document.forms[0].action | document.location |
| belge.aç | belge.URL | document.URLUnencoded | document.write |
| document.writeln | konum.hash | location.href | konum.arama |
| window.alert | window.attachEvent | window.createRequest | window.execScript |
| pencere.konum | pencere.aç | window.navigate | window.setInterval |
| window.setTimeout | XMLHTTP | | |

## C++ ve Apache'de Kod Arama

Genellikle bir C++ geliştiricisi bir web hizmeti oluştururken, b i r web sunucusu tarafından çağrılacak bir CGI programı oluşturur (ancak bu verimli değildir) veya Apache httpd çerçevesini kullanır ve HTTP isteklerini/cevaplarını işlemek için bir işleyici veya filtre yazar. Bu geliştiricilere yardımcı olmak için, bu bölümde HTTP girdi ve çıktılarını işlerken kullanılan genel C/C++ işlevleri ve işleyicilerde kullanılan bazı yaygın Apache API'leri ele alınmaktadır.

## Eski C/C++ Yöntemleri

Web istekleriyle etkileşime giren herhangi bir C/C++ kodu için, dizeleri ve çıktıları işleyen kod, mantığın herhangi bir kusur içermediğinden emin olmak için kontrol edilmelidir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| yönetici | sprint | document.referrer | fprintf |
| printf | Stdio | DOSYA | strcpy |
| strncpy | Strcat | cout | cin |
| cerr | Sistem | popen | stringstream |
| fstringstream | Malloc | ücretsiz | |

## Talep İşleme

Apache içinde kodlama yaparken, HTTP istek nesnesinden veri elde etmek için aşağıdaki API'ler kullanılabilir.

| ARANACAK DIZE | | |
|---|---|---|
| headers_in | ap_read_request | post_read_request |

## Yanıt İşleme

İstemciye gönderilecek yanıtın türüne bağlı olarak aşağıdaki Apache API'leri kullanılabilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| headers_out | ap_rprintf | ap_send_error_response | ap_send_fd |
| ap_vprintf | | | |

### Çerez İşleme

Çerez, istek başlıkları listesinden veya özel Apache işlevlerinden elde edilebilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| headers_in | headers_out | headers_out | ap_cookie_write2 |
| ap_cookie_read | ap_cookie_check_string | | |

### Günlük kaydı

Günlük mesajları, modülde bulunan özel kaydediciler (örn. log4cxx, vb.) kullanılarak, Apache tarafından sağlanan günlük API'si kullanılarak veya sadece standart çıkışa veya standart hataya yazılarak uygulanabilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| cout | cerr | ap_open_stderr_log | ap_error_log2stderr |
| ap_log_error | ap_log_perror | ap_log_rerror | |


### HTML Kodlaması

Ekip, C/C++ işleyicisinde HTML girişi veya çıkışı için bir tutamaca sahip olduğunda, HTML kodlamasını sağlamak/kontrol etmek için aşağıdaki yöntemler kullanılabilir.

| ARANACAK DIZE | | | |
|---|---|---|---|
| ap_unescape_all | ap_unescape_url | ap_unescape_url_keep2f | ap_unescape_urlencoded |
| ap_escape_path_segment | | | |

Aşağıdaki simgeler, bu kitap başlığı için başka hangi sürümlerin basılı olduğunu göstermektedir.

**ALPHA:** "Alfa Kalitesi" kitap içeriği bir çalışma taslağıdır. İçerik çok kabadır ve bir sonraki yayın aşamasına kadar geliştirme aşamasındadır.

**BETA:** "Beta Kalitesi" kitap içeriği bir sonraki en yüksek seviyedir. İçerik bir sonraki baskıya kadar hala geliştirilme aşamasındadır.

**YAYIN:** "Yayın Kalitesi" kitap içeriği, bir kitap başlığının yaşam döngüsündeki en yüksek kalite seviyesidir ve nihai bir üründür.


**Alfa**


**Beta**


**Serbest Bırak**

**ÖZGÜRSÜN:**

**Paylaşmak -** Çalışmayı kopyalamak, dağıtmak ve aktarmak

**To Remix -** eseri uyarlamak

**AŞAĞIDAKI KOŞULLAR ALTINDA:**

**Atıf**
Esere, yazar veya lisans veren tarafından belirtilen şekilde atıfta bulunmalısınız (ancak sizi veya eseri kullanımınızı onayladıklarını gösteren herhangi bir şekilde değil).

**Aynı Şekilde Paylaş**
Bu çalışmayı değiştirir, dönüştürür veya üzerine inşa ederseniz, ortaya çıkan çalışmayı yalnızca aynı, benzer veya uyumlu bir lisans altında dağıtabilirsiniz.

OWASP
Open Web Application Security Project

Açık Web Uygulama Güvenliği Projesi (OWASP), uygulama yazılımlarının güvenliğini geliştirmeye odaklanmış dünya çapında ücretsiz ve açık bir topluluktur. Misyonumuz uygulama güvenliğini "görünür" kılmaktır, böylece insanlar ve kuruluşlar uygulama güvenliği riskleri hakkında bilinçli kararlar alabilirler. Herkes OWASP'a katılmakta özgürdür ve tüm materyallerimiz ücretsiz ve açık bir yazılım lisansı altında mevcuttur.OWASP Vakfı, çalışmalarımız için sürekli kullanılabilirlik ve destek sağlayan 501c3 kar amacı gütmeyen bir hayır kurumudur.