Güvenli kodlama teknikleri ve zafiyet taramaları

**HAZIRLAYAN: Satı Doğukan Esen**

**Artificial Intelligence Security Lab (AISECLAB)**

**aiseclab.org**

**Haziran / 2023**

# **Özet**

Günümüzde dijital dünyanın hızla genişlemesiyle birlikte, güvenli yazılım geliştirme ve koruma ihtiyacın da önemi artmıştır. Geliştiricilerin, kullanıcıların verilerini ve sistemlerini korumak adına güvenlik açıklarını en aza indirmek için güvenli kodlama tekniklerini benimsemeleri lazımdır. Güvenli kodlama teknikleriyle donanmış ve zafiyet taramalarının düzenli olarak gerçekleştirildiği bir yazılım, hem güvenilirliğini artırırken hem de kullanıcıların ve verilerin korunmasını sağlama konusunda önemli bir rol oynamaktadır. Bu makalede, güvenli yazılım geliştirme yaşam döngüsünü, güvenli kodlama ilkeleri, veri doğrulama işlemleri, şifre stratejileri ve güvenli yazılım testleri açıklanmıştır.

# **Güvenli Yazılım Geliştirme Yaşam Döngüsü (Microsoft SDL)**

Microsoft geliştiricilerinin, geliştirdikleri yazılımların güvenliği için geliştirilmiş yaşam döngüsüdür. 7 aşamadan oluşmaktadır.

**Eğitim:** Güvenli tasarım, güvenli kodlama gibi temel güvenlik eğitimini içerir. Tüm çalışanlar ilk olarak güvenlik eğitiminden geçmelidir.

**Gereksinim:** Kullanıcının ihtiyaç duyduğu güvenlik sağlanır. Güvenlik gereksinimleri, kalite eşik değerleri oluşturulur. Güvenlik ve mahremiyet riski denetlenir. Geliştirme ekipleri de gereksinimleri, ürünün ele alacağı veri türünü, bilenen tehditleri, en iyi uygulamaları gibi faktörlere göre tanımlar. Güvenlik gereksinimleri tehdit ortamındaki değişiklere göre değişebilir.

**Tasarım:** Tasarım gereksinimleri oluşturulur. Saldırı yüzey analizi yapılır. Potansiyel Tehditleri riske göre tanımlamak, sınıflandırmak be derecelendirmek için tehdit modelleri oluşturulur. Tehdit modelleri, her ürünün yaşam döngüsü boyunca güncellenmelidir.

**Geliştirme:** Burada statik analiz yapılır. Onaylanmış araçlar kullanılır. Güvensiz kod kullanılmaz.

**Doğrulama:** Kodun yayınlanabilmesi için uygunluğu kontrol edilir. Manuel kod analizi, ikili analiz, bulanıklık testi gibi testler yapılır. Penetrasyon testleri uygulanır. Burada dinamik analiz yapılır. Brute force yapılır. Saldırı yüzeyi kontrol edilir.

**Sürüm:** Tüm testler geçildikten sonra uygulama hemen müşterilere sunulmaz. Belirlenmiş gruplara kademeli olarak sunulur. Olay yönetim planı oluşturulur. Son güvenlik kontrol edilir. Sürümler ve arşiv sertifikalandırılır.

**Müdehale:** Olay yönetim planı işletilir.

# **Güvenli Kodlama İlkeleri**

## Gerek duyulan en az yetkiyi verme

Bir yazılım öğesi kendisine verilen görevi tamamlarken en az yetki ile çalışmasıdır. Bu sayede uygulamanın sistemde yapabileceği değişikler sınırlanır, farklı ortamlarda çalışmaya daha kolay uyum sağlar ve bu yazılım öğesi diğer görevlerden ve öğelerden ayrı çalıştığı için daha güvenli bir sistem olur.

## Tüm erişimleri denetleme

Sistem genelinde erişim denetimi sağlanmalı ve her erişimde yetki kontrolü yapılmalıdır. Bu sayede yetkisiz erişimlerin önüne geçilir.

## Yetkileri ayırma

Sistem yüksek güvenlikli işlemler için birden fazla güvenlik koşuluna bağlı izin vermelidir. Bu sayede saldırılar ile sistemde büyük değişiklikler oluşturmak zorlaşır.

## Varsayılan değerleri güvenli hale getirme

Varsayılan olarak bırakılmış tüm değerler incelenmelidir. Bazı varsayılan değerler herkese açık olabilir. Bu durumun kısıtlanması gerekir aksi halde saldırgan deneme yanılma yöntemi ile bunları bulabilir.

## Ortak erişilebilen kaynaklara farklı kanallardan erişme

Kaynaklara erişen mekanizmalar ortak kullanılırsa bilgi sızması veya yan kanal saldırısına sebep olabilir.

## En zayıf halkayı güçlendirme

Bir yazılım en zayıf noktası kadar güvenlidir. Mutlaka en zayıf halka tespit edilmeli ve ona özel önlem alınmalıdır.

## Saldırı yüzey alanını azaltma

Yetkisiz kullanıcının, veri ile etkileşime girebileceği veya yetkisiz işlem yapılabilecek noktalardır. Bu alanı azaltmak için, gereksiz özelliklerden kaçınmalı, iletişim noktaları kısıtlanmalıdır.

## Savunma derinliği oluşturma

Bir savunma mekanizması ne kadar katmanlıysa o kadar derindir. Bu sayede olası saldırıları farklı katmanlarda işlenerek minimuma indirgenir. Her katman kendinden sorumludur. Honeypot gibi teknikler burada kullanılarak saldırgan yavaşlatılabilir.

## Anlaşılabilir, kolay ve basit güvenlik mekanizması tasarla,

Bu şekilde hata tespiti daha rahat yapılabilir. Yoksa kullanıcı güvenlik mekanizmasını aşmak için çeşitli yöntemlere başvurabilir. Bunları sağlamak için açık isimlendirilmiş fonksiyonlar, açıklama satırları gibi. Herhangi kod parçası birden fazla kez kullanılacaksa fonksiyon olmalıdır.

## Eski kod

Eski kodları yenilenecek kaynaklar hazırlanmalıdır. Yoksa eski kodların üstüne yeni sürümler geldiğinde orada güvenlik açığı olabilir. Saldırganlar ise burayı rahatça görebilir.

## Belirsizlik

Saldırganın aldığı hatalar ona öğretici yol göstermemelidir. Saldırgan bu yolla sistem hakkında bilgilenebilir. Bunun için istemciye verilen bilgiler sınırlanmalı, sınırlı geri bildirim sağlanmalıdır.

# **Veri Doğrulama ve Sınırlama**

## Tüm veri doğrulaması sunucu tarafında gerçekleşmelidir.

İstemci tarafı saldırganların müdahalesine daha açık olduğu için veriler manipüle edilebilir. Bir açık ortaya çıktığında güncellemenin tüm istemcilere yayılması gerektiğinden zorluk olabilir. Daha kritik bir zafiyet iste saldırganın istemci tarafındaki doğrulama kodunu devre dışı bırakabilmesidir.

## Sistem eğer utf-8 destekliyorsa çeviriden sonra doğrulama yapılması gerekir.

Eğer çevirmeden önce doğrulama yapılırsa sunucu bozuk karakterleri kabul edebilir, veri bütünlüğünü sağlayamayabilir. Saldırganlar bozuk veya hatalı karakterler ile verileri manipüle edebilir.

## Kullanıcıdan sağlanan tüm verileri işlemeden önce doğrulanmalıdır.

Eğer veriler doğrulanmadan işlenirse doğrulama adımı tam koruma sağlamaz. Saldırgan istediği girdileri sunucuya işletebilir. Yani kullanıcıdan alınan tüm verilere saldırgan gözüyle bakılmalıdır.

## Sistem güven sınırlarını belirlenmeli ve harici bağlantıları doğrulanamlıdır.

Sistemler farklı platform veya hizmet sağlayıcılar ile etkileşimde olduğunda harici bağlantıların güvenilirliği sağlanmalıdır. Bu bağlantıların kimlik doğrulaması, yetki kontrolü doğrulanmalıdır.

## Girdilerde whitelist kullanılmalıdır.

Whitelist kullanarak yalnızca güvenilir verilerin girişi sağlanır. Bu sayede uygulamayı zararlı girişlere karşı koruruz. Whitelist kullanırken ayrıca null karakteri, yeni satır karakteri, önceki yol karakteri gibi değerlere de dikkat etmemiz gerekir.

# **Şifre stratejileri**

## Şifre saklama

Şifreler asla plain text olarak saklanmamalıdır. Eğer plain text olarak saklanırsa sistemdeki kullanıcıları tehlikeye atar. Ayrıca kullanıcı bilgileri bulunursa kullanıcının diğer sitelerdeki üyeliklerini de tehlikeye atar.

## Şifre Değişimi istemek

Kullanıcıların şifrelerine belirli bir kullanım ömrü verip şifreleri o süre sonunda yenilemeleri istenebilir. Ayrıca yeni şifre ile önceki şifreler karşılaştırılıp benzerlik varsa yeni şifre reddedilebilir.

## Hash

Şifreleri geri dönüştürelemez olarak şifrelemeye hash denir. Bu yöntemle gerçek parola ve şifreler önce şifrelenir sonra depolanır. Hash işlemi bu bakımdan güvenli olarak gözükse de kullandığımız hash algoritmasına göre güven güvenlik derecesi değişebilir. Hash algoritmaları yavaş çalışırsa daha fazla verim alınır çünkü kırılması da yavaş olacaktır.

## Salt

Bu işlemde şifrelemeden önce parolaya ek veri eklenir. Yani saldırganın parolayı bulması için salt verisini de bilmesi gerekir. Eğer salt verisi benzersiz bir değer ise her kullanıcı için benzersiz stringler ile benzersiz saltlar oluşur. Bu işlemde şifreyi bilmek için kullanıcı adı ve salt verisini bilmek gerekir. Diğer bir salt yöntemi de random saltdır. Burada her kullanıcının hashi neredeyse rastgele ve benzersizdir.

## Şifreye gereksinim eklemek

Kullanıcıların şifrelerini güçlü yapmaları için birtakım kurallar eklenebilir. Parola uzunluğunu sınırlamadan minimum uzunluğu belirlenebilir. Parola için non-alphanumeric karakterler zorunlu tutulabilir. Kullanıcı adı ve parola arasında benzerlik olması engellenebilir. Ayrıca kullanıcı parolasının ne kadar güçlü olduğunu bilebilmelidir.

# **Güvenli Yazılım Testleri**

## Statik Analiz

Statik analizde uygulama çalıştırılmadan uygulama test edilir. Statik analizde genellikle kaynak kod incelemesi yapılır. Manuel kod incelemesine göre daha fazla kod incelenebilir ve bu işlem tekrarlanabilir. Güvenlik açıklarını bulabilirler ve bazı statik analiz programları birden fazla dili destekler. Bazıları ise IDEler ile uyumlu çalışabilir. Bu yöntemde false negative sonuçlar çıkabilir. O yüzden çıktıları bir uzmanın incelemesi gerekmektedir.

## Dinamik Analiz

Dinamik analizde ise uygulamayı test etmek için uygulamanın çalışması gerekmektedir. Dinamik analiz farklı geliştirme aşamalarını hedefler ve buna uygun çözüm sağlamaya çalışır. Ağ dinleme, ağ tarama, zayıflık taramaları kullanılır. Bazı uygulama test programlarında dinamik ve statik analiz birlikte test edilir. Statik analizde olduğu gibi dinamik analizde de false negative sonuçlar çıkabilir ve o yüzden çıktıların incelenmesi gerekmektedir.

## Manuel Kod Analizi

Manuel kod analizi zaman alıcı olsa da statik ve dinamik analizin en büyük dezavantajı olan false negative sonuçların olması ve tüm zafiyetlerin tespit edilmemesi gibi durumları engellemek için kullanılabilir. Manuel kod analizi ile güvenlik açığının önemi, nasıl değerlendirildiği ve bağlamı analiz edilebilir.

## Sızma Testi

Sızma testlerinde sisteme girdiler verilerek, sistem çıktılarına göre programda zaafiyet olup olmaması durumu test edilir. Beyaz kutu, gri kutu ve kara kutu olarak üçe ayrılır. Beyaz kutu testinde, test ekibi sistem hakkındaki tüm bilgilere sahiptir. Gri kutu testinde, test ekibi sınırlı bilgiye sahiptir. Kara kutu testinde ise test ekibi sistem hakkında bir bilgiye sahip değildir. Bu testler sonucunda ekip bir belgelendirme ile elde ettiği bilgileri firmaya sunar. Sızma testi aşamaları şu şekildedir:

**Keşif:** Pasif ve aktif şekilde sistem hakkında bilgi toplanır. Pasif bilgi toplama, sistemle etkileşime girmeden, kaynakları kullanarak sistem ile ilgili bilgi toplamaktır. Aktif bilgi toplamada ise sistemle etkileşimde bilgi toplanır. Bu bilgiler sayesinde ekip nelerle karşılaşabileceğini önceden tespit eder ve ona göre bir plan oluşturur.

**Zafiyet tarama:** Burada keşif aşamasında elde edilen bilgiler ile uygulamadaki olası zafiyetler belirlenir. Elde edilen bilgilerdeki zafiyetlerin güncelleme ile düzeltilebileceği veya hatalı olabileceği unutulmamalıdır.

## Brute force

Bu yöntem, kullanıcının erişebileceği girdi noktalarına rastgele veriler gönderilmesi ile gerçekleşir. Girdi yerleri tespit edildikten sonra girdi formatı, protokol, mesaj vb. gibi değişkenlerin uygun olanları alınır ve bunlara uygun veri kümesi oluşturulur. Bu veri kümesinin rastgele denenmesi ile işlem tamamlanır. Bu şekilde uygulamada hata olup, olmadığı öğrenilebilir.

##### Kaynakça

1. Zeng, Mini and Zhu, Feng (2021) "Secure Coding in Five Steps," Journal of Cybersecurity Education, Research and Practice: Vol. 2021 : No. 1 , Article 5.
2. Turpan, Keith (2010) “Secure Coding Practices - Quick Reference Guide” Boeing Technology | Information Technology Version 1b, Jan 2010
3. TÜBİTAK BİLGEM Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi, “GÜVENLİ YAZILIM GELİŞTİRME KILAVUZU” Rev: 1.1 28 Mayıs 2018
4. <https://learn.microsoft.com/en-us/compliance/assurance/assurance-microsoft-security-development-lifecycle>
5. <https://youtu.be/uJ7SmpkicPk>