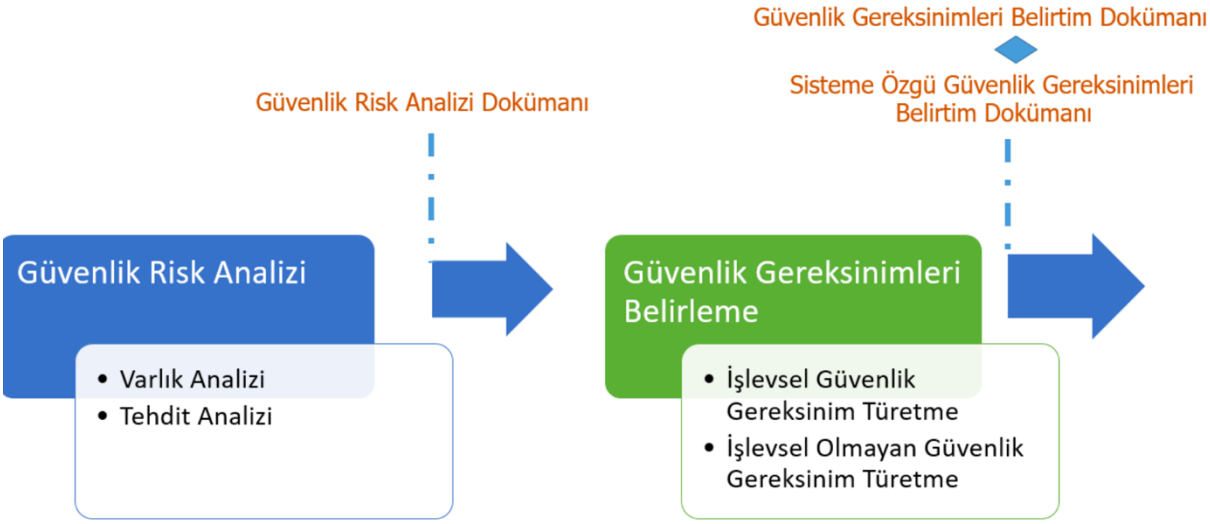
Güvenli Yazılım Geliştirme Kılavuzu

Yazılımın, saldırı veya tehdit altındayken işlevlerini doğru bir şekilde yerine getirmeye devam edecek şekilde korunmasına yazılım güvenliği denir. Bir yazılımın güvenliği yazılımı ortaya çıkarmak kadar mühimdir. Yazılım güvenliği faaliyetlerinin amacı tüm bilgi güvenliği saldırılarına karşı daha dirençli ve hatasız çalışan yazılım üretmektir. Yazılım geliştirirken proje boyunca güvenli yazılım geliştirme kurallarının nasıl ve ne zaman uygulanacağının belirlenmesi, güvenli yazılım geliştirme yaşam döngüsü faaliyetlerinin izlenmesi ve denetimi ve yazılım güvenliğinin sürekliliğinin sağlanması önem arz etmektedir. Bu sürecin yürütülmemesi durumunda saldırılara maruz kalabilir ve yazılımın tamamen durdurulması ya da doğru çalışmasının engellenmesi, istenmeyen amaca hizmet etmesi veya bilgi sızıntılarına engel olunamaz.
Tasarımı gizli tutarak güvenlik yaklaşımı (Security through obscurity), yeni nesil araç ve yöntemlerle (tersine mühendislik, kod analizi araçları vb.) artık geçerliliğini yitirmiştir. Güvenlik tasarımının karşı tarafça bilindiği varsayılarak güvenlik analizlerinin yapılması gerekmektedir. Bu bağlamda, uygulamanın tehdit modeli tanımlanmalı, güvenlik tasarım doğrulama yöntemleri (tehdit modelleme ve risk analizi, biçimsel yöntemler, tasarım gözden geçirme vb.) projelerde standart olarak uygulanmalıdır.

Güvenli Yazılım Geliştirme Yaşam Döngüsü;

Güvenli bir yazılımın tasarımı, gerçeklenmesi, yapılandırılması, kurulması ve desteklenmesi, yazılımın birçok saldırıya karşın doğru bir şekilde çalışmaya devam edecek, karşı koyamadığı saldırıların etkisini sınırlamak için gerekli önlemleri içinde barındıracak ve kısa sürede normal çalışmaya devam edecek şekilde olabilmesini sağlayacak bir süreçte hayata geçirilir.
Yazılım ve çevresindeki ortam arasındaki etkileşimlerden dolayı ortaya çıkabilecek zafiyetleri en aza indirmek için, geçmiş zafiyetlerin incelenmesi, dış inceleme, sızma (penetrasyon) testi ve olay müdahalesi gibi önlemler uygulanır. Ayrıca geliştiriciler güvenli yazılım geliştirme hususunda eğitilmeli ve güvenlik gereksinimlerinin tanımlamaları da yapılmalıdır.



Yazılım geliştirme sürecinde bulunan varlık türlerine örnek olarak şunlar verilebilir:
Bilgi varlıkları – İş Kuralları – Servisler veya metotlar – Yazılım kodu – Firmaya özgü formüller – Şifreleme yöntem ve anahtarları – Veritabanları – Kişiler veya kişilerin sahip olduğu belirli bilgi ve yetenekler – Hesap bilgileri ve hesapla ilişkilendirilmiş fonlar – İş kayıtları

Varlıkların tespit edilmesinde Veri Akış Şemalarının (Data-Flow Diagram) oluşturulması yönetimi uygulanabilir. Bu yöntemde süreçler, harici öğeler, veri depoları, veri akışları ve güven sınırları belirlenir. Varlıklar tespit edildikten sonra Varlık Değerlendirme (Asset Valuation) adımı gerçekleştirilir. Bu adımda matris, Delphi yöntemi, Lineer değerlendirme (ISO 27005) gibi yöntemler izlenebilir. Matris örneği olarak:

Maddi değer? İş sürecine etkisi?	Düşük	Orta	Yüksek
Düşük	(Genel) Bilgi varlıkları	İş kayıtları	Hesap bilgileri
Orta	İş Kuralları	Yazılım Kodu	Kişiler Müşteri veritabanları
Yüksek	Kullanıcı bilgileri veritabanı	Yazılım Servisleri Şifreleme yöntem ve anahtarları	Firmaya özgü formüller

Daha sonra tehdit profili değerlendirme analizi yapılır.

Kaynak ---------- Yetenek/Teknik Zorluk	Düşük	Orta	Yüksek
Düşük	Meraklı çocuk	Siber savaşçı	Siber terörist
Orta	Çevrimiçi sosyal hacker	Siber suçlu	Kötü niyetli geliştirici/operatör
Yüksek	İç tehdit	Hacker organizasyonları	Siber casusluk

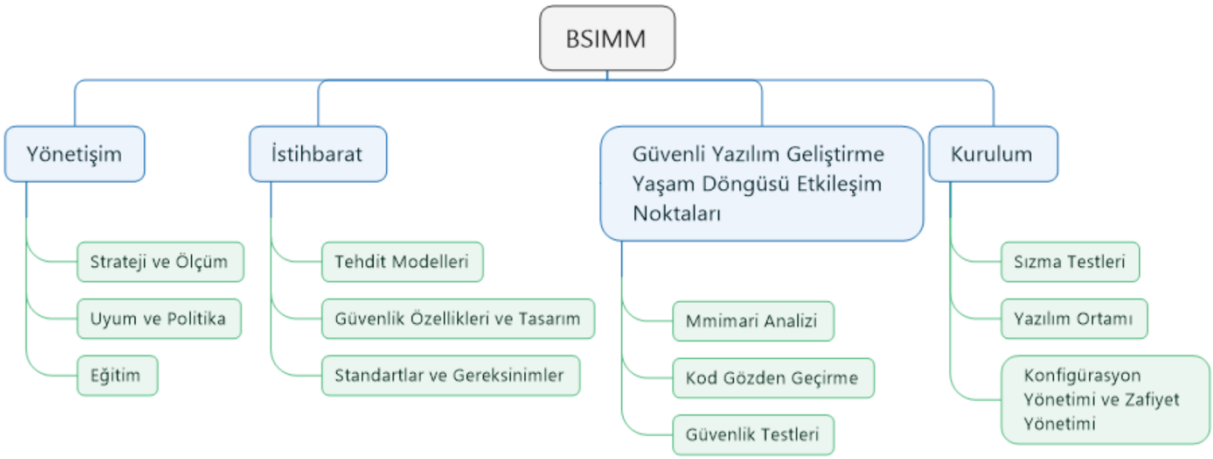
Son olarak ise buradan, etki ve olasılık değerleri riski hesaplamak için kullanılabilir. Bu amaçla aşağıdaki formüller örnek olarak kullanılabilir.

- Olasılık = Zayıflık * Tehdit derecesi
- Etki = MAX (Varlığın kuruma olan değeri, Saldırının dolaylı etkileri)
- Risk = Olasılık * Etki

Güvenli yazılım geliştirme olgunluk modelleri bir kuruluşun yazılım güvenliği pratiklerini ne derecede uyguladığını ortaya koyan derecelendirmeye dayalı ölçüm modelleridir.
OWASP tarafından ortaya konulmuş olan SAMM modeli ve Building Security In Maturity Model (BSIMM) modeli bulunmaktadır ve bunlar aşağıda özetlenmiştir.

SAMM;

BSIMM;



Yazılımın güvenliğini sağlamak için belirlenmiş bazı yöntemler mevcuttur ve bunlar;

Statik analiz ve sembolik koşturma yöntemleri için yazılımın çalıştırılan halini yansıtan bir modeli veya benzetimini oluşturarak, bilinen yazılım hata türleri bu model üzerinde aranır.

Statik Analiz bir yazılımın belli başlı özelliklerini onu çalıştırmadan analiz etmeyi amaçlayan yöntemlerden biridir. Statik analiz ile yazılımları karşılaştırırken hangi tür yazılım hatalarını çözebildikleri harici kaynaklardan doğrulanmalıdır. Örneğin Carnegie Mellon Üniversitesi'nin https://www.securecoding.cert.org sitesinde C/C++, Java gibi dillere ilişkin yazılım kodlama hataları ve bu hataların hangi araçlarla ne oranda bulunabildiğine ilişkin detaylı incelemeler mevcuttur. Statik analizde Boon ve Find Security Bugs gibi araçlar kullanılabilir.

Sembolik Koşturma kodun derilerindeki gizli hataların bulunması için kullanılır. Bu yöntemde yazılımın kodu üzerinde bir soyutlama yapılarak tüm olası çalıştırmalar modellenir. Yazılım testi kullanılarak sadece tanımlanan akış kapsamındaki hatalar bulunabilir.

Dinamik analiz, negatif ve rastgele test yöntemlerinde ise doğrudan yazılımın koşturulması sırasında gerçek ortamın benzetimi oluşturulur ve yazılım bazı hatalara zorlanır. Ardından hatalar analiz edilir ve sonuçlara ulaşılmaya çalışılır.

Dinamik Analiz uygulamadaki açıklıkları tespit etmek için çalışan yazılım üzerinde ve çalışma ortamında gerçekleştirilen analiz araç ve yöntemleridir. Dinamik analiz için kullanılan uygulamalara örnek olarak Nmap, Wireshark, sqlmap verilebilir.

Negatif Test hatalı veya güvenlik kurallarına uymayan durumların ve istisnaların yazılım tarafından kontrol edildiğinin test edilmesine denir. Örneğin; en az 8 karakterlik bir parola gerektiren bir sayfanın 7 karakterli parola ile test edilmesi ve sonuçta yazılımın bu hususta hata mesajı vermesi.

Rastgele Test bir yazılıma geçersiz, beklenmeyen veya rastgele veri girdisinde bulunarak istenmeyen davranışta bulunup bulunmayacağının tespit edilmesine denir.

Biçimsel Program Analizi statik analizlerin matematik ve mantık temelli olarak gerçekleştirilmesine denir.

Model Doğrulayıcılar yazılımda mevcudiyeti tespit edilmesi istenen gizlilik, kimlik doğrulama, erişim denetimi, bütünlük gibi güvenlik özelliklerinin bir yazılımda mevcut olup olmadığının doğrulanması için kullanılan yöntemdir.

Mantık Yöntemleri kodun içerisinde yapılan varsayımların ve beklenen sonuçların mantıksal ifadelerle tanımlanmasından sonra otomatik doğrulayıcı araçlar yardımıyla beklenen sonuçların doğrulanması yöntemidir.

Model Tabanlı Geliştirme yazılımın güvenlik davranışı uygulama alanına özel üst seviye bir dille (Domain Specific Language-DSL) tanımlanarak kaynak kodun büyük bir kısmının veya tamamının oluşturulan modelden üretilmesi yöntemidir. Daha çok gömülü sistemlerde uygulanan bir yöntemdir. Kullanıcı arayüzü yoğun yazılımlar için bu yöntemin kullanımı uygun değildir.

Doğrulanmış Araç ve Kod Depoları Kullanma yazılımcılar genellikle ihtiyaç duydukları genel amaçlı işlemler için (sıralama, veritabanı erişimi vb.) internette paylaşılan kod parçacıklarını kullanmaktadır ve çoğunlukla da bu kod parçacıkları herhangi bir güvenlik testine veya gözden geçirmeye tabi tutulmadan yazılımın kod deposuna eklenmektedir. Dolayısıyla güvenilir olduğundan emin olmadığımız kütüphaneleri ve kod parçacıklarını kullanmamamız gerekir. Doğrulanmış kütüphanelerin oluşturulması çabalarına bir örnek olarak, OWASP ESAPI projesi verilebilir.

Kod Güçlendirme önceden geliştirilmiş bir yazılımın güvenlik bakış açısı ile kaynak kodlarının, yapılandırma dosyalarının, derleyicisinin, dinamik kütüphanelerin ve özel amaçlı diğer kütüphanelerin biçimsel yöntemlerle doğrulanması ve güçlendirilmesidir.

İspat Taşıyan Kod uygulamanın çalıştırılabilir kodunu takip ederek biçimsel yöntemlerle analiz ederek kodun güvenli şekilde çalışıp çalışmayacağına karar veren yöntemdir.

Sızma Testi yöntem olarak kara kutu, gri kutu ve beyaz kutu yaklaşımları kullanılmaktadır. Beyaz kutu yaklaşımında kaynak kod dahil sistem hakkında tüm bilgilere sahipken, gri kutu yaklaşımında ise kapsam hakkında sınırlı bilgi ile sızma testi gerçekleştirilir. Kara kutu yaklaşımında test edilen sistem hakkında hiçbir ön bilgi bulunmadığı kabul edilmektedir ve ayrıca testi gerçekleştiren ekibin kaynak koda erişimi bulunmamaktadır.
Sızma testinde öncelikle keşif yapılır ve bu aşamada pasif ve aktif bilgi toplama aşamaları bulunur. Keşif aşamasında Cain & Abel, Nmap, Scapy, Wireshark gibi programlar kullanılabilir. Keşif aşamasından sonra ise zafiyet taraması yapılır, burada keşiften toplanan bilgilerden yararlanılarak keşifte bir zafiyet olasılığı bulunmuşsa o zafiyet sömürülmeye çalışılır. Bu aşamada zafiyet testi için Metasploit kullanılabilir.

Fuzz Testi ile uygulamanın girdi noktalarına beklenmedik rasgele veriler sistematik olarak gönderilerek uygulamanın ürettiği hatalar gözlemlenir. Öncelikle hedef yazılım üzerinde testi gerçekleştirecek girdi noktalarının tespiti edilir. Bu tespit yapıldıktan sonra istenen verinin formatına uygun olarak uygun veri kümesi oluşturulur ve uygulamada kullanılır. Ardından sistemin tepkisi incelenir ve son aşamada hata oluşturan girdiler tespit edilerek bulunan hataların güvenlik açısından sömürülebilir (exploitable) olup olmadığı tespit edilir. Fuzz testi için OWASP WSFuzzer ve Wfuzz kullanılabilir.

Yazılım Çalışma Ortamı Güvenliğini Sağlayan Yöntemler şunlardır;

Bazı güvenlik işlevlerinin sistem tarafında ortak kullanılabilir mekanizmalarla sağlanması, saldırılara karşı dayanıklı ve güvenlik hizmetlerini varsayılan olarak sağlayan çalışma ortamı oluşturulması, yazılımın güvenliğini artıracak önlemler arasındadır.

Uygulama Konteyneri (Application Container) Kullanımı üzerinde çalıştığı bilgisayarın bazı kaynaklarını, kendi içerisinde çalışan uygulama veya sistem için yalıtan bir nesnedir. Konteynırlar, yeterince yalıtıldığında yazılım zayıflıklarını ciddi şekilde azaltabilmektedir.

Mikro Servis Mimarisi İle Yazılımların Geliştirilmesi yazılımın küçük servisler halinde ve her bir servisin kendi süreci içerisinde birbirleri ile konteynerler ile iletişim kuracak şekilde tasarlanmasına mikro servis mimarisi denir. Bu mimari ile tasarlanan bir yazılımda yer alan her bir servis de mikro servis olarak adlandırılır. Her mikro servis iş veya süreç gerçekleştirir.

Sanallaştırma Ortamlarının Kullanılması tek bir donanım ya da fiziksel sunucu üzerinde birden çok program yığınının (computing stack) sanallaştırmaya özgü bazı yazılımlar ve donanımlar sayesinde çalıştırılmasını sağlayan bir çözümdür.

Yazılım Güvenliğini Sağlamada Diğer Yöntemler şunlardır;

Programlama Dilleri ile Güvenliğinin Artırılması yazılımın kendinden beklenenleri tam anlamıyla yapmasının dışında yapması istenmeyen şeyleri de ihtimali ne kadar küçük olursa olsun gerçekleşmesini engelleyebilmesi gerekmektedir. Her dil için o dile uygun güvenli kodlama standartı oluşturulmalıdır.

Değişen Hedef Savunmasının Uygulanması bir yazılımın normal çalışmasını bozmadan, yapısının ve özelliklerinin sürekli olarak değiştirilerek saldırganın zayıflıkları sömürme olasılığı en aza indirilmeye çalışılmasına değişen hedef savunması denir. Buna örnek olarak bellek taşımı saldırılarını engellemek için kullanılan ve yığın belleğini rastgele hale getiren ASLR yöntemi gösterilebilir.

Ağ ve Sistem Seviyesinde Önlem Alınması her ne kadar yazılım geliştirme sürecinin parçası olmasa da yazılımın koştuğu ortamdaki ağ ve sistem seviyesinde önlemlerin alınması, yazılımdaki açıklıkların sömürülmesini zorlaştıran bir unsur olduğundan göz ardı edilmemelidir. Gözden geçirme, sızma testi, sıkılaştırma yöntemleri ile testler yapılabilir.

Güvenli yazılım geliştirmede yazılım güvenliğinin ilkeleri bazı standart kuruluşlar tarafından belirlenmiştir ve bunlar harmanlanmış bir özet biçiminde kabaca şunlardır;

- Her bir prosedür, modül kendi işini bitirmesi için gerekli en az haklarla çalıştırılmalı ve bu süreç esnasında bu haklara gereken en az süre boyunca sahip olmalı
- Her bir nesneye yapılan erişimlerde yetki kontrolü yapılmalı ve bu kontrol normal durumların dışında başlatma, kapatma veya istek, yanıt aşamalarında da yapılmalı
- Yüksek güvenlik gerektiren işlemlerde tek bir koşula bağlı olarak izin verilmemeli
- Yazılımda kullanılan tüm varsayılan değerler güvenliği arttıracak şekilde seçilmeli ve her bir nesnenin varsayılan erişimi hiç olmalı
- Sistem ve yazılım içerisindeki modüller arasında yalıtım sağlanmalı
- Yazılımın güvenliği o yazılımdaki en zayıf halkanın güvenliği kadardır. Bu halka tespit edilerek yeniden tasarlanıp gerçeklenmeli
- Gereksiz özellikler eklenmemelidir
- Yetkisiz bir kullanıcının yazılım ortamından veri alabileceği, girebileceği veya yetkisiz işlemlerde bulunabileceği noktalar tespit edilmeli ve bu noktalar izlenmeli, sınırlandırılmalıdır
- Savunma mekanizmaları peş peşe uygulanmalıdır
- Güvenlik mekanizmaları mümkün olduğunda basit olmalıdır ki bu şekilde hata oluşturduğunda yetkililer tarafından kolay anlaşılmalı ve düzeltilebilmeli
- Kullanıcılar işini en kolay şekilde ancak en az yetkiyle yapabilmeli
- Uygulamanın güvenlik katmanları birbirinden ayrılmalı ve güvenlik işlevlerinin normal işlemlerin gerçekleştirildiği çalışma ortamından farklı bir izole ortam içerisinde gerçekleştirilmesi sağlanmalıdır.
- Uygulama veritabanında kişisel veri içeren birincil anahtar (kimlik no, e-posta adresi vb.) kullanılmamalıdır.
- Sistem mimarisi zayıf yönleri veya zayıf noktaları bulmak için saldırganın bakış açısı ile incelenmeli.
- Kaynak kodun zafiyet ve faaliyet analizleri yapılmalı.
- Güvenli diller, güvenli ve onaylanmış kütüphaneler kullanılmalıdır. Eski ve güvensiz kütüphaneler kullanılmamalıdır.
- Kasıtlı veya kasıtsız uygulamada güvenlik açığı yatacak kodu kimin yarattığının kontrolü için sürüm kontrolü yapılmalıdır.
- Kod kalitesi ölçümünde pep8 vb. kodlama stilleri üzerinden kontroller yapılır. Bu kontroller kodlamanın yazım biçimlerini, kod içinde bulunan boşluklar, değişken isimleri, fonksiyonların girdi sayılarını, satır sayılarını ve benzeri özellikleri kontrol eder.
- Otomatik araçlar ürettiği hata çıktılarının bir kısmı yanlış alarm (false pozitif) hatalardır. Bundan dolayı otomatik araçların çıktıları manuel olarak da incelenmelidir.
- Koruma gerektiren kritik verinin tam olarak korunabilmesi için, aynı zamanda uygulama kaynak kodu, binary kodu ve çalışma zamanı kodunun da korunması gerekmektedir.
- Mimarideki tüm yazılım bileşenleri tanımlı olmalı ve ihtiyaç duyulmayan bileşenler kaldırılmalıdır.
- Uygulama saldırıya uğradığında izlenecek saldırı karşılık planı oluşturulmalıdır.
- Periyodik olarak veritabanları, uygulamalar ve uygulama verisi yedeklemesi yapılmalıdır.

- Gereksinimler açık, tutarlı, tam, uygulanabilir, takip edilebilir ve doğrulanabilir olmalıdırlar.
- Tüm parola alanlarında kullanıcı giriş yaparken kullanıcının parolası maskelenmeli ve açık olarak görünmemelidir.
- Kimlik doğrulama başarısız olduğu takdirde güvenli bir duruma geçilmeli ve saldırganların yetkisiz oturum açmaları engellenmelidir.
- Hesaba yeniden erişebilecek tüm hesap kimlik doğrulama işlevleri (profil güncelleme, parolamı unuttum vb.) en az ana kimlik doğrulama mekanizması kadar saldırılara dayanıklı olmalıdır.
- Hassas işlevler gerçekleştirilmeden önce, yeniden kimlik doğrulama, daha güçlü bir mekanizmayla kimlik doğrulama, ikinci faktör veya işlem imzalama gibi yöntemler uygulanmalıdır.
- Kaynak kodunda veya kaynak kodu depolarında gizli bilgiler, API anahtarları ve parolalar mevcut olmamalıdır.
- Uygulamanın yönetim arayüzlerine güvenilmeyen taraflarca erişilmesi engellenmelidir.
- Uygulama kullanıcının son başarılı oturum açma tarih ve saatini görüntülemelidir.
- Güvenilmeyen kaynaklardan alınan dosyaların türü doğrulanmalı ve zararlı bir içeriğe sahip olup olmadığı kontrol edilmelidir.
- Oturumlar belirli bir süre etkinlik olmadığında kendiliğinden sonlanmalıdır.
- Kimlik doğrulamayla erişilen tüm sayfalardan oturum kapatma işlevine erişilebilmelidir.
- Oturum kimliğinin URL, hata mesajları ve iz kayıtları içerisinde yer almaması sağlanmalıdır. URL içerisinde oturum kimliğinin yeniden yazılması engellenmelidir.
- Tüm kimlik doğrulama ve yeniden kimlik doğrulama işlemleri sonucunda yeni bir oturum ve yeni bir oturum kimliği üretilmelidir.
- Kullanıcı sadece yetkilendirildiği uygulama bileşenlerine ve kaynaklara erişebilmeli ve bunları kullanabilmelidir.
- Bellekte tutulan önemli veriler gereksinimi sona erdiğinde güvenlik ihlali oluşturamayacak şekilde silinmelidir.
- Uygulama, hassas veri ve kişisel verileri içeren hata mesajı veya iz kaydı üretmemelidir.
- Uygulama tarafından, istemci ve sunucu tarafında, kabul edilen her bir veri tipi için girdi doğrulama denetimi yapılmalıdır.
- HTML form alanlarının veri girdileri, REST çağrıları, HTTP üst başlıkları, çerezler, toplu işlem dosyaları, RSS beslemeleri gibi veri girdileri için doğrulama denetimi yapılmalıdır.
- Uygulama, sunucu ve istemci tarafında dil kodlaması (encoding) saldırılarına karşı dayanıklı olmalıdır.
- Uygulama, kişisel verileri şifreli olarak saklamalı ve bu verilerin taşınmasında korumalı iletişim kanallarını kullanmalıdır.
- Uygulamanın istemci tarafında çalışan kodları, kişisel verileri başka ortamlara aktarmamalı (konsola yazma, başka dosya olarak kaydetme, yerel veya uzak uygulamalara transfer etme vb.), güvensiz ortamlarda (ortak dizin, USB disk vb.) güvensiz yöntemlerle (açık metin olarak, zayıf şifreleme algoritma kullanarak şifreleme vb.) saklamamalıdır.
- Kullanılan veritabanının dışarıya aktarımı ancak veritabanı yönetim yetkisi olan hesaplarla yapılmalı ve öncesinde veritabanındaki kişisel verilerin silinmesi sağlanmalıdır.
- Değişen parola fonsiyonu eski parolayı, yeni parolayı ve bir parola onayını kapsamalıdır.
- Kaba kuvvet saldırıları ya da servis dışı bırakma saldırıları gibi otomatik yapılan yaygın kimlik doğrulama saldırılarını önlemek için istekler azaltılmalıdır. Aşırı kimlik doğrulama denemelerini engellenmeli.
- Parolalar için bir en uzun geçerlilik süresi tanımlanmış olmalıdır.
- Uygulama, ayar ve denetim dosyaları kullanıcı verisiyle aynı konumda depolamamalıdır.

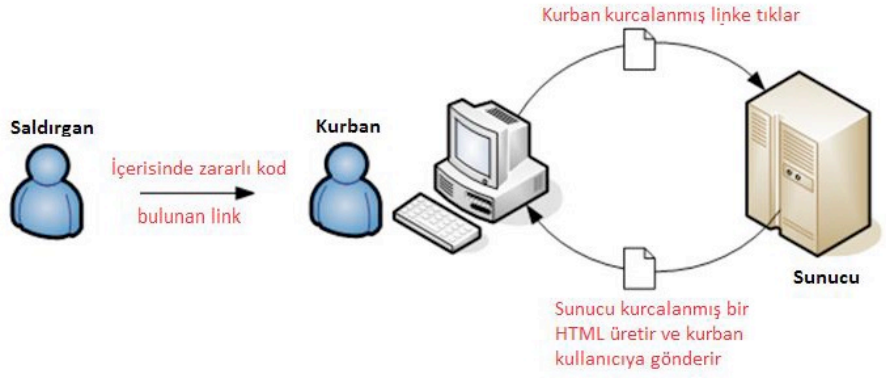
- Desteklenmeyen, güvensiz veya teknolojisi zaman aşımına uğramış istemci teknolojileri kullanılmamalıdır.
- Uygulama tarafından etkin ve aynı zamanlı oturumların sayısı sınırlandırılabilmelidir. Sızma testi ile test edilebilir.
- Her bir kullanıcının uygulamadaki etkin oturumları görüntülenebilmeli, kullanıcı herhangi bir etkin oturumunu sonlandırabilmelidir.
- Web uygulamalarında oturum çerezlerinde HTTPOnly bayrağı etkin olmalıdır.
- Uygulama, başarısız sistem başlatma, başarısız sonlandırma veya başarısız kapatma gibi işlemlerde güvenli bir duruma geçmelidir. Negatif test ile test edilebilir.
- Tüm kriptografik modüllerin, güvenli bir şekilde hataya düştüğü doğrulanmalıdır. Hata yönetimi "Oracle Padding" atağına imkan tanımayacak şekilde olmalıdır. Negatif test ile test edilebilir.
- Tüm rastgele üretilen sayılar, dosya isimleri, global eşsiz değerler (GUID) ve karakter dizilerinin saldırgan için tahmin edilemez olması sağlanmalıdır. Rastgele sayıların yüksek entropiye sahip olarak üretilmelidir. Negatif test ile test edilebilir.
- Tüm anahtar ve şifreler kullanımları tamamlandığında, tamamen sıfırlanarak yok edilmelidir. Negatif test ile test edilebilir.
- Tüm formlarda istemci tarafında yapılan ön bellekleme işlevselliği önemli veriler için kapatılmalıdır.
- Siteler arası komut dosyası çalıştırma (XSS) engellemede uygulama-istemci arasındaki hassas veri iletişim için HTTP başlığı veya gövdesi kullanılmalıdır. Postman, Wireshark vs. ile test edilebilir.
- Uygulama, saklama gereksinimi sona erdikten sonra önemli verileri güvenlik sonunu yaratmayacak şekilde silinmelidir.
- Sunucuya gelen isteklerin öngörülmeyen bir sayıda ya da büyüklükte olup olmadığı kontrol edilebilmelidir. Negatif test ile test edilebilir.
- İz kayıtlarında olayların zaman sıralamasına ilişkin araştırma yapılabilecek şekilde zaman bilgisi yer almalıdır.
- Uygulama tarafından üretilen iz kayıtları hassas bilgi içermemelidir.
- Uygulama hassas bilgileri formlarda bulunan gizli alanlarda saklamamalıdır.
- Uygulama Cross-Site Request Forgery (CSRF)'dan kaynaklanan açıklıklardan korunma mekanizmasına sahip olmalıdır. Fuzz testi, kaynak kod zayıflık analizi ile test edilebilir.
- Uygulamanın çalışma ortamı, bellek taşması saldırılarına dayanıklı olmalıdır veya mevcut güvenlik mekanizmaları bellek taşmasını engellemelidir.
- Sunucuda yapılan girdi doğrulama hataları, isteğin reddi ile sonuçlanmalı ve iz kaydı oluşturulmalıdır.
- SQL Injection engellemek için bütün veritabanı sorguları, parametre olarak yapılmalı ve veritabanına erişimde kullanılan dile karşı önleyecek denetimler yapılmalıdır. Veritabanı açıklık tarama aracı, web uygulama zayıflık tarayıcı veya negatif test ile test edilebilir.
- LDAP-Active Directory Injection için uygulama, yetki onaylama hizmetlerinin enjeksiyonu açıklıklarını önleyici güvenlik denetimlerini yapmalıdır. Tasarım gözden geçirme, işlevsel test, negatif test ve kaynak kod gözden geçirme ile test edilebilir.
- İşletim sistemi komut enjeksiyonu için uygulama, işletim sistemi komut enjeksiyonu açıklıklarını önleyici güvenlik denetimlerini yapmalıdır. Kaynak kodu açıklık tarayıcısı, Web uygulama açıklık tarayıcısı, uygulama açıklık tarayıcısı, tasarım gözden geçirme, işlevsel test, negatif test, kaynak kod gözden geçirme ile test edilebilir.
- Uygulama, girdi alınan içerik bir dosya yolu içeriyorsa, uzak ya da yakın dosya içerme açıklıklarını önleyici güvenlik denetimlerini yapmalıdır.

- Uygulama, XML açıklıklarını (XPath sorgu saldırıları, XML harici öğe saldırıları, XML enjeksiyonu vb.) önleyici güvenlik denetimlerini yapmalıdır.
- Uygulama, web servislerini iyi yapılandırılmış en az TLS v1.2 veya SSL gibi muadil güvenlik önlemi sunan bir protokol ile sunacak şekilde tasarlanmalıdır.
- Uygulama, web servis kimlik doğrulama ve yetkilendirmesi için oturum temelli yapılar kullanacak şekilde tasarlanmalıdır.
- Uygulama, web servisi ile gönderilen veride betik (script) içermeyecek şekilde tasarlanmalıdır. Statik analiz ile test edilebilir.
- Uygulama, web servislerinden şifreli olarak paylaşılan verileri yine şifreli olarak saklayacak şekilde tasarlanmalıdır.
- Uygulama, kişisel veriler üzerinde işlem yapılması ana amaç olmayan durumlarda kişisel verileri maskeleyerek görüntülemeli, aktarmalı veya işlemelidir.

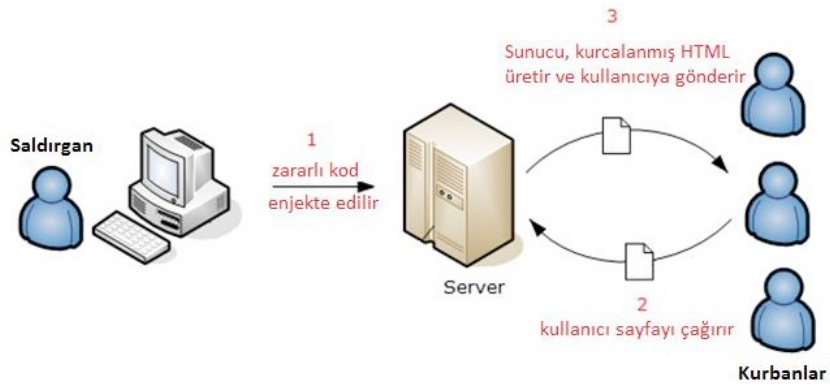
Ek olarak web uygulama güvenliği konusunda üstünde durulan kritik kavramlar ve bunların detaylı açıklamaları aşağıdadır:

Cross Site Scripting (XSS) bir tarayıcıda izinsiz olarak kod çalıştırmaktır. 3 tipi vardır ve bunlar:

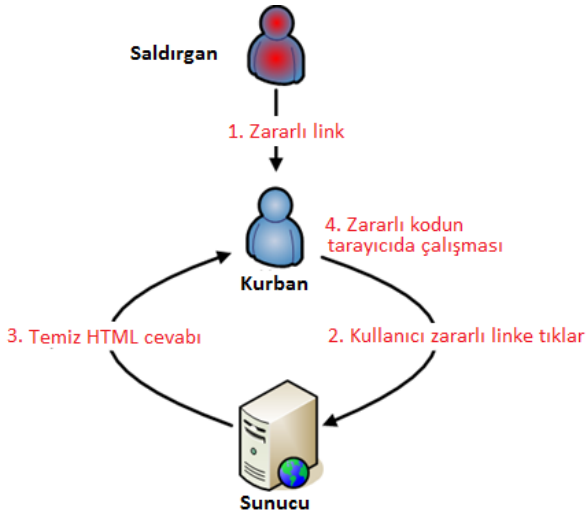
Yansıtılan XSS(Reflected XSS) saldırgan zararlı bir link hazırlar ve kurban kullanıcıyı bu linke tıklamaya ikna eder. Kurban kullanıcının tarayıcısından zararlı bir talep gönderilir. Linkteki zararlı HTML, web sayfası içerisine eklenerek kullanıcıya geri gönderilir. Kurban kullanıcının tarayıcısı zararlı HTML'i çalıştırır.



Depolanan XSS(Stored XSS) saldırgan zararlı kodunu veritabanı veya sunucuya enjekte eder. Sayfayı ziyaret eden tüm kullanıcıların tarayıcıları bu zararlı kodu çalıştırır.



Dom Tabanlı XSS(DOM Based XSS) kullanıcıya gönderilen HTML zararlı bir kod içermez fakat tarayıcı DOM objesini çağırır ardından DOM objesi zararlı kodu çalıştırır.



SQL Enjeksiyonu(SQLi) veritabanında izinsiz SQL sorgusu çalıştırmaktır. Kullanıcıdan alınan girdinin SQL sorgusu olarak işlenmesi sonucu ile gerçekleşir.

SELECT name FROM products WHERE id=‘100’; yerine
SELECT name FROM productsWHERE id=‘100’ UNION SELECT passwordFROM usersWHERE userid=‘1’ gibi bir sorgu çalıştırılması ile olur.

SQL enjeksiyonuna karşı en etkili önlem string eklemedir. Örneğin;
String username = Session["username"];
String password = Session[“password"];
String selectCommand = "SELECT * FROM usersWHERE username='" + username + "' AND password='" + password + "'";
SqlCommand myCommand = new SqlCommand(selectCommand, DataConnection);
SqlDataReader dr = myCommand.ExecuteReader();
Şeklinde bir kullanımla SQLi önüne geçilebilir.

KAYNAKÇA

NIST Secure Software Devolopment Framework
TUBITAK Güvenli Yazılım Geliştirme Kılavuzu
TUBITAK SGEP Web Uygulama Güvenliği