2. Introduction (Giriş)

2.1 The OWASP Testing Project (OWASP Test Projesi)

OWASP Test Projesi uzun yıllardır geliştirilmektedir. Projenin amacı, insanların *whatwhywhenwhere*web uygulamalarının ne, neden, ne zaman, nerede ve *nasıl* test edildiğini anlamalarına yardımcı olmaktır. Proje, yalnızca basit bir kontrol listesi veya ele alınması gereken sorunların reçetesi değil, tam bir test çerçevesi sundu. Okuyucular bu çerçeveyi kendi test
programlarını oluşturmak veya başkalarının süreçlerini belirlemek için bir şablon olarak kullanabilirler. Test Kılavuzu, hem genel test çerçevesini hem de çerçeveyi pratikte uygulamak için gereken teknikleri ayrıntılı olarak açıklamaktadır.

Test Kılavuzunu yazmak zor bir iş olduğunu kanıtlamıştır. İnsanların kılavuzda açıklanan kavramları uygulamalarına izin veren ve aynı zamanda kendi çevrelerinde ve kültürlerinde çalışmalarını sağlayan fikir geliştirmek ve içerik geliştirmek bir meydan okumaydı. Ayrıca, web uygulama testinin odağını penetrasyon testinden yazılım geliştirme yaşam döngüsüne entegre edilmiş testlere dönüştürmek de zordu.

Ancak grup, projenin sonuçlarından çok memnun. Dünyanın en büyük şirketlerinden bazılarında yazılım güvenliğinden sorumlu olan birçok endüstri uzmanı ve güvenlik uzmanı, test çerçevesini doğruluyor. Bu çerçeve, kuruluşların güvenilir ve güvenli yazılımlar oluşturmak için web uygulamalarını test etmelerine yardımcı olur. Çerçeve sadece zayıflık alanlarını vurgulamıyor, ancak bu kesinlikle OWASP rehberlerinin ve kontrol listelerinin çoğunun bir yan ürünüdür. Bu nedenle, belirli test tekniklerinin ve teknolojilerinin uygunluğu hakkında zor kararlar verilmelidir. Grup, herkesin bu kararların hepsine katılmayacağını tam olarak anlıyor. Bununla birlikte, OWASP, fikir birliğine ve deneyime dayalı farkındalık ve eğitim yoluyla zaman içinde yüksek zemini ve değişimi kültürü alabilir.

Bu kılavuzun geri kalanı aşağıdaki gibi düzenlenir: bu giriş, web uygulamalarının test edilmesinin ön koşullarını ve test kapsamını kapsar. Ayrıca başarılı test ve test

teknikleri, raporlama için en iyi uygulamalar ve güvenlik testi için iş davaları ilkelerini de kapsar. Bölüm 3, OWASP Test Çerçevesini sunar ve yazılım geliştirme yaşam döngüsünün çeşitli aşamaları ile ilgili teknik ve görevlerini açıklar. Bölüm 4, kod muayenesi ve penetrasyon testi ile belirli güvenlik açıkları (örneğin, SQL Enjeksiyonu) için nasıl test edilir.

Measuring Security: the Economics of Insecure Software (Önleyici Güvenlik: Güvensiz Yazılımın Ekonomisi)

Yazılım mühendisliğinin temel bir ilkesi, Tom DeMarco tarafından Controlling Software Projects: Management, Measurement ve Tahminler'den bir alıntıda özetlenmiştir:

> Ölçülemeyeceğiniz şeyleri kontrol edemezsiniz.

Güvenlik testi de farklı değil. Ne yazık ki, güvenliği ölçmek çok zor bir süreçtir.

Vurulması gereken bir husus, güvenlik ölçümlerinin hem belirli teknik konularla (örneğin, belirli bir güvenlik açığının ne kadar yaygın olduğu) hem de bu sorunların yazılım ekonomisini nasıl etkilediği ile ilgili olduğudur. Çoğu teknik insan en azından temel sorunları anlayacaktır veya kırılganlıkları daha iyi anlayabilir. Ne yazık ki, çok azı bu teknik bilgiyi parasal terimlere çevirebilir ve güvenlik açığının potansiyel maliyetini uygulama sahibinin işine ölçebilir. Bu gerçekleşene kadar, CIO'lar güvenlik yatırımı konusunda doğru bir getiri geliştiremez ve daha sonra yazılım güvenliği için uygun bütçeler ataymayacaktır.

Güvensiz yazılımın maliyetinin göz korkutucu bir görev görünebileceğini tahmin etmek göz korkutucu bir görev görünse de, bu yönde önemli miktarda çalışma olmuştur. 2018 yılında BT Yazılımı Kalitesi Konsorsiyumu summarized özetlendi:

> ABD'de 2018'de kalitesiz yazılımların maliyeti yaklaşık 2,84 trilyon dolar...

Bu belgede açıklanan çerçeve, insanları tüm gelişim süreci boyunca güvenliği ölçmeye teşvik ediyor. Daha sonra güvensiz yazılımın maliyetini, işletme üzerindeki etkisiyle ilişkilendirebilir ve sonuç olarak uygun iş süreçleri geliştirebilir ve riski yönetmek için kaynak atayabilirler. Web uygulamalarını ölçme ve test etmenin

diğer yazılımlardan daha da kritik olduğunu unutmayın, çünkü web uygulamaları İnternet üzerinden milyonlarca kullanıcıya maruz kalır.

What is Testing? (Test nedir ?)

Bir web uygulamasının gelişim yaşam döngüsü sırasında birçok şeyin test edilmesi gerekir, ancak test aslında ne anlama gelir? The Oxford İngilizce Sözlüğü "test" olarak tanımlar:

> Test (noun): bir şeyin kalitesini, performansını veya güvenilirliğini belirlemeyi amaçlayan bir prosedür, özellikle de yaygın kullanıma alınmadan önce.

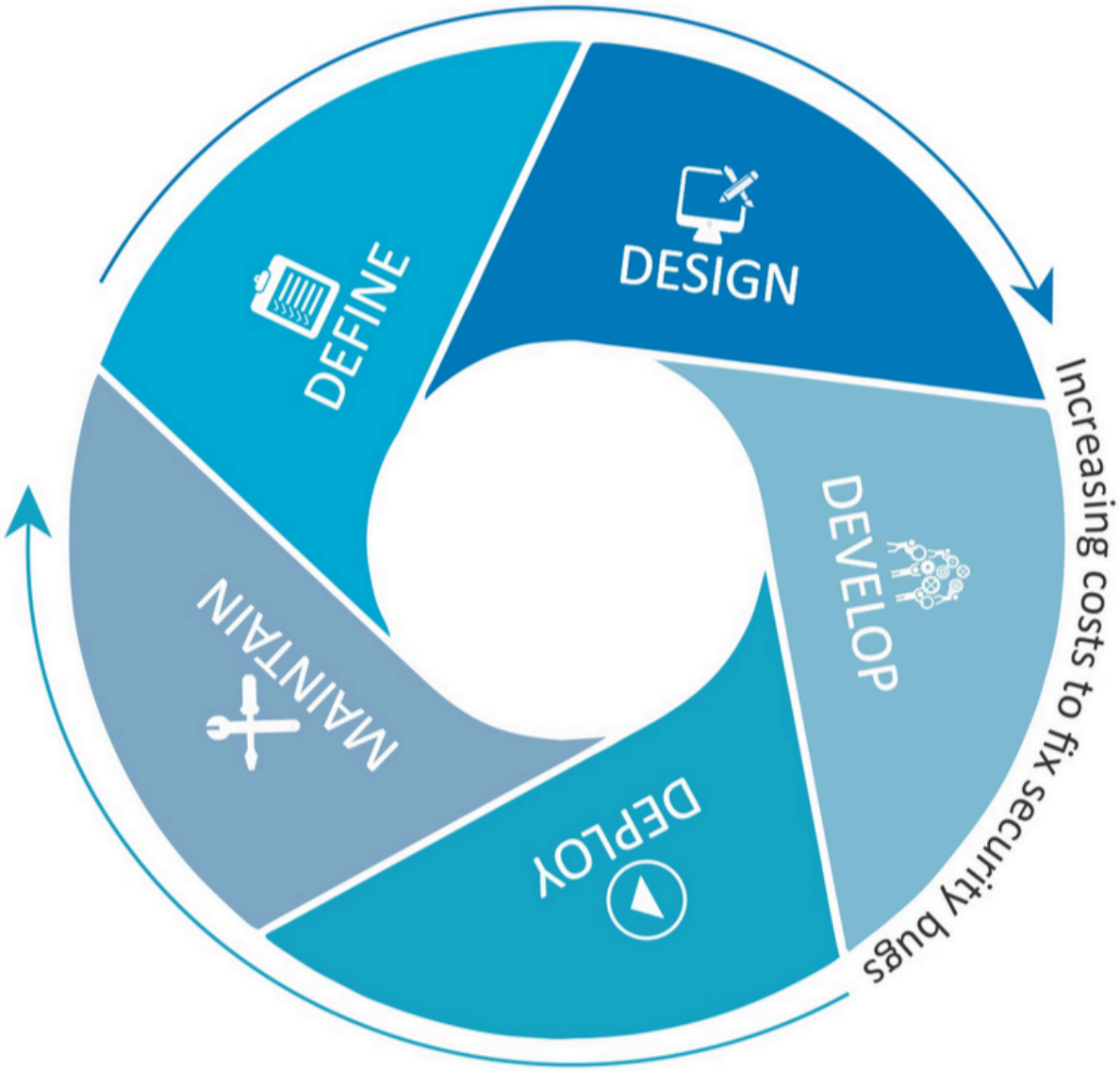
Bu belgenin amaçları doğrultusunda, test, bir sistemin durumunu veya başvuruyu bir dizi kritere kıyasla karşılaştırma işlemidir. Güvenlik endüstrisinde, insanlar sık sık iyi tanımlanmış veya tamamlanmış olan bir dizi zihinsel kritere karşı test yaparlar. Bunun bir sonucu olarak, birçok yabancı güvenlik testini siyah bir sanat olarak görüyor. Bu belgenin amacı bu algıyı değiştirmek ve derinlemesine güvenlik bilgisi olmayan kişilerin testte bir fark yaratmasını kolaylaştırmaktır.

Why Perform Testing? (Neden Test Yapıyor?)

Bu belge, kuruluşların bir test programını neyin içerdiğini anlamalarına yardımcı olmak ve modern bir web uygulama test programı oluşturmak ve işletmek için yapılması gereken adımları belirlemelerine yardımcı olmak için tasarlanmıştır. Kılavuz, kapsamlı bir web uygulama güvenlik programı yapmak için gereken unsurların geniş bir görünümünü sunar. Bu kılavuz, mevcut uygulamalar ve endüstri en iyi uygulamaları arasındaki boşluğu belirlemeye yardımcı olmak için referans ve bir metodoloji olarak kullanılabilir. Bu kılavuz, kuruluşların kendilerini endüstri akranlarıyla karşılaştırmalarına, yazılımı test etmek ve sürdürmek için gereken kaynakların büyüklüğünü anlamalarını veya bir denetime hazırlanmalarını sağlar. Bu bölüm, bir uygulamanın nasıl test edileceğinin teknik ayrıntılarına girmez, çünkü amaç tipik bir güvenlik organizasyon çerçevesi sağlamaktır. Bir başvurunun nasıl test edileceğine dair teknik ayrıntılar, bir penetrasyon testi veya kod incelemesinin bir parçası olarak, bu belgenin kalan bölümlerinde kapsanacaktır.

When to Test? (Ne zaman test edilir?)

Günümüzde çoğu insan, daha önce oluşturulana ve yaşam döngüsünün dağıtım aşamasında olana kadar yazılımı test etmez (yani, kod oluşturulmuş ve çalışan bir web uygulamasına anında alınmıştır). Bu genellikle çok etkisiz ve maliyet-yasaklayıcı bir uygulamadır. Güvenlik hatalarının üretim uygulamalarında görünmesini önlemek için en iyi yöntemlerden biri, her bir aşamaya güvenlik dahil ederek Yazılım Geliştirme Yaşam Döngüsünü (SDLC) geliştirmektir. Bir SDLC, yazılım eserlerinin geliştirilmesine dayatılan bir yapıdır. Bir SDLC şu anda ortamınızda kullanılmıyorsa, bir tane seçmenin zamanı geldi! Aşağıdaki rakam, jenerik bir SDLC modelinin yanı sıra, böyle bir modelde güvenlik hatalarını sabitlemenin (tahmini) artan maliyetini göstermektedir.



Şekil 2-1: Jenerik SDLC Modeli

Şirketler, güvenliğin geliştirme sürecinin ayrılmaz bir parçası olmasını sağlamak için genel SDLC'lerini denetlemelidir. SDLC'ler, güvenliğin yeterince kaplandığından ve geliştirme süreci boyunca kontrollerin etkili olmasını sağlamak için güvenlik testlerini içermelidir.

What to Test? (Ne test edilir?)

Yazılım geliştirmeyi insanların, süreçlerin ve teknolojinin bir kombinasyonu olarak düşünmek yararlı olabilir. Yazılımı "yaratıl" faktörlerse, bunların test edilmesi gereken faktörler olması mantıklıdır. Günümüzde çoğu insan genellikle teknolojiyi veya yazılımın kendisini test eder.

Etkili bir test programı aşağıdakileri test eden bileşenlere sahip olmalıdır:

- **İnsanlar** – yeterli eğitim ve farkındalık olmasını sağlamak;
- **Süreç** – yeterli politika ve standartlar olduğundan ve insanların bu politikaları nasıl izleyeceklerini bilmelerini sağlamak;
- **Teknoloji** – sürecin uygulanmasında etkili olmasını sağlamak.

Bütünsel bir yaklaşım benimsenmedikçe, bir uygulamanın teknik uygulamasının test edilmesi, mevcut olabilecek yönetim veya operasyonel güvenlik açıklarını ortaya çıkarmaz. İnsanları, politikaları ve süreçleri test ederek, bir kuruluş daha sonra teknolojideki kusurlara kendini gösterecek sorunları yakalayabilir, böylece böcekleri erken ortadan kaldırabilir ve kusurların temel nedenlerini tanımlayabilir. Aynı şekilde, bir sistemde bulunabilecek teknik sorunların yalnızca bir kısmını test etmek, eksik ve yanlış bir güvenlik duruşu değerlendirmesine neden olacaktır.

Fidelity National Financial'da Bilgi Güvenliği Başkanı Denis Verdon, New York'taki OWASP AppSec 2004 Konferansı'nda bu yanlış anlama için mükemmel bir benzetme sundu:

> Arabalar uygulamalar gibi inşa edilmiş olsaydı ... güvenlik testleri sadece ön etkiyi üstlenirdi. Arabalar test edilmeyecek veya acil manevralarda stabilite, fren etkinliği, yan darbe ve hırsızlığa karşı direnç için test edilmeyecektir.

How To Reference WSTG Scenarios (WSTG Senaryolarına Nasıl Atıfta Bulunulur)

Her senaryonun formatta bir tanımlayıcısı vardır `WSTG-<category>-<number>` "Kategori", test veya zayıflık türünü tanımlayan 4 karakterli bir üst kasa dizisidir ve 'sayı' 01'den 99'a kadar sıfır yastıklı bir sayısal değerdir. Örneğin: `WSTG-INFO-02` İkinci Bilgi Toplama testidir.

Tanımlayıcılar sürümler arasında değişebilir, bu nedenle diğer belgelerin, raporların veya araçların formatı kullanması tercih edilir: `WSTG-<version>-<category>-<number>`, Nerede: 'verim' noktalama işareti kaldırılmış versiyon etiketidir. Örneğin: `WSTG-v42-INFO-02` Özellikle 4.2 sürümden ikinci Bilgi Toplama testi anlamına gelir.

Tanımlayıcılar dahil olmadan kullanılırsa `<version>` Daha sonra en son Web Güvenlik Test Kılavuzu içeriğine atıfta bulundukları varsayılmalıdır. Açıkçası, rehber büyüdükçe ve değiştikçe bu sorunlu hale gelir, bu yüzden yazarlar veya geliştiriciler sürüm öğesini içermelidir.

Linking (Bağlantı)

Web Güvenlik Test Kılavuzu senaryolarına bağlantı verme, sürüme edilmiş bağlantılar kullanılarak yapılmalıdır `stable` ya da `latest` Zamanla kesinlikle değişecektir. Ancak, proje ekibinin tanımlı bağlantıların değişmemesidir. Örneğin: `https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server.html`. . Not: The `v42` element 4.2 versiyonunu ifade eder.

Feedback and Comments (Geri bildirim ve yorumlar)

Tüm OWASP projelerinde olduğu gibi, yorumları ve geri bildirimleri memnuniyetle karşılıyoruz. Özellikle çalışmalarımızın kullanıldığını ve etkili ve doğru olduğunu bilmek isteriz.

2.2 Principles of Testing (Test Prensipleri)

Yazılımda güvenlik hatalarını bulmak için bir test metodolojisi geliştirirken bazı yaygın yanlış anlamalar vardır. Bu bölüm, profesyonellerin yazılım üzerinde güvenlik testleri yaparken dikkate alması gereken bazı temel ilkeleri kapsar.

There is No Silver Bullet (Gümüş mermi yok)

Bir güvenlik tarayıcısının veya uygulamalı bir güvenlik duvarının saldırıya karşı birçok savunma sağlayacağını veya çok sayıda sorunu tanımlayacağını düşünmek cazip olsa da, gerçekte güvensiz yazılım sorununa gümüş mermi yoktur. Uygulama güvenliği değerlendirme yazılımı, düşük asılı meyve bulmak için ilk geçiş olarak

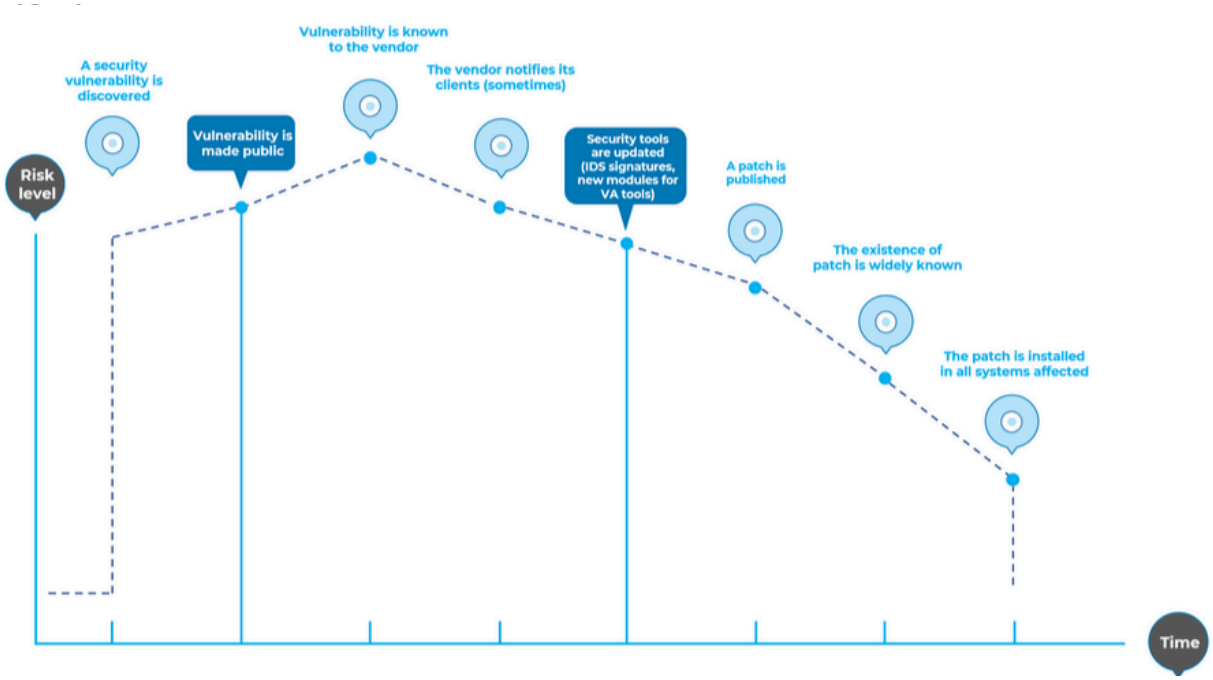
yararlı olsa da, genellikle önemsiz ve derinlemesine değerlendirmede etkisizdir veya yeterli test kapsamı sağlar. Unutmayın ki güvenlik bir ürün değil bir süreçtir.

There is No Silver Bullet (Stratejik Olarak Düşünün, Taktiksel Değil)

Güvenlik uzmanları, 1990'larda bilgi güvenliğinde yaygın olan yama ve nüfuz modelinin safsatasını fark ettiler. Yama ve penetra modeli, bildirilen bir hatayı düzeltmeyi içerir, ancak kök nedenin uygun şekilde araştırılmamasız. Bu model genellikle aşağıdaki figürde gösterilen maruz kalma penceresi olarak da adlandırılan güvenlik açığı penceresi ile ilişkilidir. Dünya çapında kullanılan ortak yazılımlardaki güvenlik açıklarının evrimi, bu modelin etkisizliğini göstermiştir. Maruz kalma
pencereleri hakkında daha fazla bilgi için, Güvenlikte Schneier'e bakın.

Symantec'in İnternet Güvenliği Tehdit Raporu gibi güvenlik açığı çalışmaları, dünya çapındaki saldırganların reaksiyon süresiyle, tipik kırılganlık penceresinin yama kurulumu için yeterli
zaman sağlamadığını göstermiştir, çünkü bir güvenlik açığının ortaya çıkarılması ve geliştirilmesine ve serbest bırakılmasına karşı otomatik bir saldırı arasındaki sürenin her yıl azaldığını göstermiştir.

Yama ve penetra modelinde birkaç yanlış varsayım vardır. Birçok kullanıcı, yamaların normal işlemlere müdahale ettiğine veya mevcut uygulamaları bozabileceğine inanmaktadır. Tüm kullanıcıların yeni piyasaya sürülen yamalardan haberdar olduğunu varsaymak da yanlıştır. Sonuç olarak, bir ürünün tüm kullanıcıları yamaların yazılımın nasıl çalıştığına müdahale edebileceği veya yamanın varlığı hakkında bilgi sahibi olmadıkları için yama uygulayacaktır.



Şekil 2-2: Güvenlik Açığı Penceresi

Bir uygulama içinde tekrarlanan güvenlik sorunlarını önlemek için Yazılım Geliştirme Yaşam Döngüsü'ne (SDLC) güvenlik oluşturmak esastır. Geliştiriciler, geliştirme metodolojisine uygun ve çalışan standartlar, politikalar ve yönergeler geliştirerek SDLC'ye güvenlik oluşturabilirler. Tehdit modellemesi ve diğer teknikler, en çok risk altında olan bir sistemin parçalarına uygun kaynakların atamaya yardımcı olmak için kullanılmalıdır.

The SDLC is King (SDLC Kraldır)

SDLC, geliştiriciler tarafından iyi bilinen bir süreçtir. Güvenliği SDLC'nin her aşamasına entegre ederek, organizasyon içinde halihazırda yürürlükte olan prosedürlerden yararlanan uygulama güvenliğine bütünsel bir yaklaşım sağlar. Bir kuruluşun kullandığı SDLC'nin her kavramsal aşamasının bir kuruluşun kullandığı SDLC'nin her kavramsal aşamasının (yani, tanımlamak, tasarlamak, geliştirmek, dağıtmak) çeşitli aşamaların birer kuruntuya bağlı olarak değiştirilebileceğini unutmayın. Her aşama, uygun maliyetli ve kapsamlı bir güvenlik programı sağlamak için mevcut sürecin bir parçası olması gereken güvenlik hususlarına sahiptir.

Varoluşta hem tanımlayıcı hem de kuralcı tavsiyeler sağlayan birkaç güvenli SDLC çerçevesi vardır. Bir kişinin tanımlayıcı veya kuralcı tavsiye alıp almadığı SDLC

işleminin olgunluğuna bağlıdır. Esasen, kuralcı tavsiye, güvenli SDLC'nin nasıl çalışması gerektiğini gösterir ve tanımlayıcı tavsiyeler gerçek dünyada nasıl kullanıldığını gösterir. İkisinin de yeri var. Örneğin, nereden başlayacağınızı bilmiyorsanız, kuralcı bir çerçeve SDLC içinde uygulanabilecek potansiyel güvenlik kontrollerinden oluşan bir menü sağlayabilir. Tanımlayıcı tavsiyeler daha sonra diğer kuruluşlar için iyi çalışan şeyleri sunarak karar sürecini yönlendirmeye yardımcı olabilir. Tanımlayıcı güvenli SDLC'ler BSIMM'yi içerir; ve kuralcı güvenli SDLC'ler OWASP'nin Açık Yazılım Güvencesi Olgunluk Modeli (OpenSAMM) ve ISO / IEC 27034 Parça 1-7, hepsi yayınlandı (4. bölüm hariç).

Test Early and Test Often (Erken Test ve Sık Sık Test Edin)

Bir hata SDLC içinde erken tespit edildiğinde, daha hızlı ve daha düşük bir maliyetle ele alınabilir. Bir güvenlik böceği bu konuda işlevsel veya performansa dayalı bir hatadan farklı değildir. Bunu mümkün kılmanın önemli bir adımı, kalkınmayı ve QA ekiplerini ortak güvenlik sorunları ve bunları tespit etmenin ve önlemenin yolları hakkında eğitmektir. Her ne kadar yeni kütüphaneler, araçlar veya diller daha az güvenlik hatasına sahip programların tasarlanmasına yardımcı olsa da, yeni tehditler sürekli ortaya çıkıyor ve geliştiriciler, geliştirdikleri yazılımı etkileyen tehditlerin farkında olmalıdır. Güvenlik testinde eğitim, geliştiricilerin bir uygulamayı bir saldırganın bakış açısından test etmek için uygun zihniyeti edinmelerine yardımcı olur. Bu, her kuruluşun güvenlik sorunlarını mevcut sorumluluklarının bir parçası olarak değerlendirmesine izin verir.

Test Automation (Test Otomasyonu)

Modern gelişim metodolojilerinde (ancak bunlarla sınırlı olmamak üzere): çevik, devops / devsecops veya hızlı uygulama geliştirme (RAD) dikkate alınmalıdır. Temel güvenlik bilgilerini / analizini korumak ve “düşük asılı meyve” türü zayıflıklarını belirlemek için sürekli entegrasyon / sürekli dağıtım (CI / CD) iş akışlarına entegre güvenlik testlerine alınmalıdır. Bu, dinamik uygulama güvenlik testlerinden (DAST), statik uygulama güvenlik testi (SAST) ve yazılım kompozisyonu analizi (SCA) veya standart otomatik serbest bırakma iş akışları sırasında veya düzenli olarak planlanmış bir temelde bağımlılık izleme araçlarından yararlanarak yapılabilir.

Understand the Scope of Security (Güvenliğin kapsamını anlayın)

Belirli bir projenin ne kadar güvenlik gerektireceğini bilmek önemlidir. Korunacak olan varlıklara nasıl ele alınacağını belirten bir sınıflandırma verilmelidir (örneğin,

gizli, gizli, çok gizli).
Tartışmalar, belirli bir güvenlik gereksiniminin karşılanmasını sağlamak için yasal konsey ile gerçekleşmelidir. ABD'de, gereksinimler Gramm-Leach-Bliley Yasası veya Kaliforniya SB-1386 gibi eyalet yasalarından federal düzenlemelerden gelebilir. AB ülkeleri merkezli kuruluşlar için hem ülkeye özgü düzenleme hem de AB Direktifleri geçerli olabilir. Örneğin, 96/46/EC4 ve Yönetmelik (AB) 2016/679 (Genel Veri Koruma Yönetmeliği) başvuruda bulunan başvurularda kişisel verilerin işlenmesini zorunlu kılar.

Develop the Right Mindset (Doğru Zihniyeti Geliştirin)

Güvenlik açıkları için bir başvuruyu başarılı bir şekilde test etmek, "kutunun dışında" düşünmeyi gerektirir. Normal kullanım durumları, bir kullanıcı beklenen şekilde kullandığında uygulamanın normal davranışını test eder. İyi güvenlik testi, beklenenin ötesine geçmeyi ve uygulamayı kırmaya çalışan bir saldırgan gibi düşünmeyi gerektirir. Yaratıcı düşünme, hangi beklenmedik verilerin bir uygulamanın güvensiz bir şekilde başarısız olmasına neden olabileceğini belirlemeye yardımcı olabilir. Ayrıca, web geliştiricileri tarafından yapılan ve her zaman doğru olmayan varsayımları ve bu varsayımların nasıl altüst edilebileceğini bulmaya yardımcı olabilir. Otomatik araçların güvenlik açıkları için test yapmanın zayıf bir işi yapmasının bir nedeni, otomatik araçların yaratıcı düşünmemesidir. Yaratıcı düşünme, çoğu web uygulaması benzersiz bir şekilde geliştirildiği için (ortak çerçeveleri kullanırken bile) geliştirilmiştir.

Understand the Subject (Konuyu anlayın)

Herhangi bir iyi güvenlik programındaki ilk büyük girişimlerden biri, uygulamanın doğru belgelenmesini gerektirmelidir. Mimari, veri akışı diyagramları, kullanım durumları vb. resmi belgelerde kaydedilmeli ve gözden geçirilmeye hazır hale getirilmelidir. Teknik şartname ve uygulama belgeleri, yalnızca istenen kullanım durumlarını değil, aynı zamanda özel olarak izin verilmeyen herhangi bir kullanım vakasını da listeleyen bilgileri içermelidir. Son olarak, bir kuruluşun uygulamalarına ve ağına (örneğin, saldırı tespit sistemleri) yönelik saldırıların izlenmesine ve eğilimini sağlayan en azından temel bir güvenlik altyapısına sahip olmak iyidir.

Use the Right Tools (Doğru Aletleri Kullanın)

Zaten gümüş mermi aleti olmadığını belirtmiş olsak da, araçlar genel güvenlik programında kritik bir rol oynamaktadır. Birçok rutin güvenlik görevini otomatikleştirebilecek bir dizi Açık Kaynak ve ticari araç vardır. Bu araçlar, güvenlik

personeline görevlerinde yardımcı olarak güvenlik sürecini basitleştirebilir ve hızlandırabilir. Bununla birlikte, bu araçların tam olarak ne yapabileceğini ve yapamayacağını anlamak önemlidir, böylece aşırı satılmaz veya yanlış kullanılmaz.

The Devil is in the Details (Şeytan Ayrıntılar İçinde)

Bir uygulamanın yüzeysel bir güvenlik incelemesini yapmamak ve tamamlandığını düşünmek kritik öneme sahiptir. Bu, ilk etapta bir güvenlik incelemesi yapmamış gibi tehlikeli olabilecek yanlış bir güven duygusu aşılayacaktır. Bulguları dikkatlice gözden geçirmek ve raporda kalabilecek herhangi bir yanlış pozitif ayıklamak hayati önem taşır. Yanlış bir güvenlik bulgusunun bildirilmesi genellikle bir güvenlik raporunun geri kalanının geçerli mesajını baltalayabilir. Uygulama mantığının mümkün olan her bölümünün test edildiğini ve her kullanım durumu senaryosunun olası güvenlik açıkları için araştırıldığını doğrulamak için dikkatli olunmalıdır.

Use Source Code When Available (Mevcut olduğunda Kaynak Kodunu Kullanın)

Siyah kutu penetrasyon test sonuçları, bir üretim ortamında güvenlik açıklarının nasıl ortaya çıktığını göstermek için etkileyici ve yararlı olabilirken, bir uygulamayı güvence altına almanın en etkili veya verimli yolu değildir. Dinamik testin tüm kod tabanını test etmesi zordur, özellikle de birçok yuvalanmış koşullu ifade varsa. Başvuru için kaynak kodu mevcutsa, incelemelerini yaparken onlara yardımcı olmak için güvenlik personeline verilmelidir. Uygulama kaynağındaki açıkları bir kara kutu katılımı sırasında kaçırılacak güvenlik açıklarını keşfetmek mümkündür.

Develop Metrics (Metrikleri Geliştirin)

İyi bir güvenlik programının önemli bir parçası, işlerin iyiye gidip gitmediğini belirleme yeteneğidir. Test angajmanlarının sonuçlarını izlemek ve organizasyon içindeki uygulama güvenlik eğilimlerini ortaya çıkaracak metrikler geliştirmek önemlidir.

İyi metrikler gösterecektir:

- Daha fazla eğitim ve eğitim gerekiyorsa;
- Geliştirme ekibi tarafından açıkça anlaşılamayan belirli bir güvenlik mekanizması varsa;
- Eğer bulunan toplam güvenlikle ilgili sorun sayısı azalıyorsa.

Mevcut kaynak kodundan otomatik olarak oluşturulabilecek tutarlı metrikler, kuruluşun yazılım geliştirmede güvenlik hatalarını azaltmak İçin getirilen

mekanizmaların etkinliğini değerlendirmesinde de yardımcı olacaktır. Metrikler kolayca geliştirilmez, bu nedenle IEEE tarafından sağlanan standart gibi bir standart kullanmak iyi bir başlangıç noktasıdır.

Document the Test Results (Test Sonuçlarını Belgeleyin)

Test sürecini sonuçlandırmak için, hangi test işlemlerinin yapıldığına, hangi test işlemlerinin yapıldığına, ne zaman yapıldığına ve test bulgularının ayrıntılarının resmi bir kaydını üretmek önemlidir. Geliştiriciler, proje yönetimi, işletme sahipleri, BT departmanı, denetim ve uyumu içerebilecek tüm ilgili taraflar için yararlı olan rapor için kabul edilebilir bir format üzerinde anlaşmak akıllıca olacaktır.

Rapor, maddi risklerin var olduğu işletme sahibine açıkça tanımlamalı ve bunu daha sonraki hafifletme eylemleri için desteklerini almak için yeterli bir şekilde yapmalıdır. Rapor ayrıca, geliştiricinin anlayacağı bir dilde sorunları çözmek için güvenlik açığından ve ilgili önerilerden etkilenen kesin işlevi belirlemede geliştiriciye açık olmalıdır. Rapor ayrıca başka bir güvenlik test cihazının sonuçları yeniden üretmesine izin vermelidir. Raporun yazılması, güvenlik testçisinin kendilerine aşırı yükseğe yüklenmemelidir. Güvenlik testçileri genellikle yaratıcı yazma becerileri ile ünlü değildir ve karmaşık bir rapor üzerinde anlaşmak, test sonuçlarının düzgün bir şekilde belgelenmediği durumlara yol açabilir. Bir güvenlik testi raporu şablonu kullanmak zamandan tasarruf edebilir ve sonuçların doğru ve tutarlı bir şekilde belgelenmesini ve izleyici için uygun bir formatta olmasını sağlayabilir.

2.3 Testing Techniques Explained (Açıklanan Test Teknikleri)

Bu bölüm, bir test programı oluştururken kullanılabilecek çeşitli test tekniklerinin üst düzey bir genel bakışını sunar. Bu teknikler için belirli metodolojiler sunmaz, çünkü bu bilgiler Bölüm 3'te kapsanmaktadır. Bu bölüm, bir sonraki bölümde sunulan çerçeve için bağlam sağlamak ve dikkate alınması gereken bazı tekniklerin avantajlarını veya dezavantajlarını vurgulamak için dahil edilmiştir. Özellikle şunu önleyeceğimizi de ele alacağız:

- Manuel Denetimler ve Değerlendirmeler
- Tehdit Modellemesi
- Kod İncelemesi

- Penetrasyon Testi

2.4 Manual Inspections and Reviews (Manuel Denetimler ve Değerlendirmeler)

Overview (Genel Bakış)

Manuel denetimler, tipik olarak insanların, politikaların ve süreçlerin güvenlik etkilerini test eden insan incelemeleridir. Manuel denetimler, mimari tasarımlar gibi teknoloji kararlarının incelenmesini de içerebilir. Genellikle belgeleri analiz ederek veya tasarımcılar veya sistem sahipleriyle röportaj yaparak yürütülürler.

Manuel denetimler ve insan incelemeleri kavramı basit olsa da, mevcut en güçlü ve etkili teknikler arasında olabilir. Birine bir şeyin nasıl çalıştığını ve neden belirli bir şekilde uygulandığını sorarak, test cihazı herhangi bir güvenlik endişesinin belirgin olup olmadığını hızlı bir şekilde belirleyebilir. Manuel denetimler ve incelemeler, yazılım geliştirme yaşam döngüsü sürecini test etmenin ve yeterli bir politika veya becerinin yerleştirilmesini sağlamanın birkaç yolundan biridir.

Hayattaki birçok şeyde olduğu gibi, manuel denetimler ve incelemeler yaparken, güven-ama doğrusal bir modelin benimsenmesi önerilir. Testçinin gösterildiği veya söylendiği her şey doğru olmayacaktır. Manuel incelemeler, insanların güvenlik sürecini anlayıp anlamadığını, politikadan haberdar edilmiş olup olmadığını ve güvenli uygulamaları tasarlamak veya uygulamak için uygun becerilere sahip olup olmadığını test etmek için özellikle iyidir.

Belgeleri manuel olarak gözden geçirme, güvenli kodlama politikaları, güvenlik gereksinimleri ve mimari tasarımlar da dahil olmak üzere diğer faaliyetlerin tümü manuel denetimler kullanılarak gerçekleştirilmelidir.

Advantages (Avantajları)

- Destekleyici bir teknoloji gerekmez
- Çeşitli durumlara uygulanabilir
- Esnek
- Takım çalışmasını teşvik eder
- SDLC'nin başlarında

Disadvantages (Dezavantajları)

- Zaman alıcı olabilir
- Her zaman mevcut olmayan destekleyici malzeme
- Önemli insan düşüncesi ve becerisinin etkili olmasını gerektirir

2.5 Threat Modeling (Tehdit Modellemesi)

Owerview (Genel Bakış)

Tehdit modellemesi, sistem tasarımcılarının sistemlerinin ve uygulamalarının karşılaşabileceği güvenlik tehditleri hakkında düşünmelerine yardımcı olmak için popüler bir teknik haline geldi. Bu nedenle, tehdit modellemesi uygulamalar için risk değerlendirmesi olarak görülebilir. Tasarımcının potansiyel güvenlik açıkları için hafifletme stratejileri geliştirmesini sağlar ve kaçınılmaz olarak sınırlı kaynaklarını ve dikkatini en çok ihtiyaç duyan sistemlere odaklamalarına yardımcı olur. Tüm uygulamaların geliştirilmiş ve belgelenmiş bir tehdit modeline sahip olması önerilir. SDLC'de tehdit modelleri mümkün olduğunca erken oluşturulmalı ve uygulama geliştikçe ve gelişme ilerledikçe tekrar gözden geçirilmelidir.

Bir tehdit modeli geliştirmek için, risk değerlendirmesi için NIST 800-30 standardını takip eden basit bir yaklaşım benimsemenizi öneririz. Bu yaklaşım şunları içerir:

- Uygulamayı ayrıştırmak – uygulamanın nasıl çalıştığını, varlıklarını, işlevselliğini ve bağlantısını anlamak için manuel muayene işlemi kullanın.
- Varlıkların tanımlanması ve sınıflandırılması - varlıkları somut ve somut olmayan varlıklarla sınıflandırın ve iş önemine göre sıralayın.
- Teknik, operasyonel veya yönetsel olsun, potansiyel güvenlik açıklarını keşfetmek.
- Potansiyel tehditleri keşfetmek – tehdit senaryolarını kullanarak veya ağaçlara saldırarak bir saldırganın bakış açısından potansiyel saldırı vektörlerine gerçekçi bir bakış açısı geliştirin.
- Hafifletme stratejileri oluşturmak – gerçekçi olduğu düşünülen tehditlerin her biri için hafifletici kontroller geliştirin.

Bir tehdit modelinden elde edilen çıktı, değişiklik gösterebilir, ancak tipik olarak listelerin ve diyagramların bir koleksiyonudur. Uygulamanın tasarımında potansiyel

güvenlik kusurları için test uygulamaları için referans olarak kullanılabilecek çeşitli Açık Kaynak projeleri ve ticari ürünler uygulama tehdit modelleme metodolojilerini destekler. Tehdit modelleri geliştirmek ve uygulamalarda bilgi risk değerlendirmeleri yapmak için doğru veya yanlış bir yol yoktur.

Advantages (Avantajları)

- Sistemin Pratik Saldırgan Görünümü
- Esnek
- SDLC'nin başlarında

Disadvantages (Dezavantajları)

- İyi tehdit modelleri otomatik olarak iyi bir yazılım anlamına gelmez

2.6 Source Code Review (Kaynak Kod İncelemesi)

Owerview (Genel Bakış)

Kaynak kodu incelemesi, güvenlik sorunları için bir web uygulamasının kaynak kodunu manuel olarak kontrol etme işlemidir. Birçok ciddi güvenlik açığı başka bir analiz veya test türü ile tespit edilemez. Popüler deyişin dediği gibi “gerçekten neler olduğunu bilmek istiyorsanız, doğrudan kaynağa gidin.” Neredeyse tüm güvenlik uzmanları, koda gerçekten bakmanın yerini almadığı konusunda hemfikirdir. Güvenlik sorunlarını tanımlamak için tüm bilgiler kodda, bir yerde var. İşletim sistemleri gibi kapalı yazılımın test edilmesinin aksine, web uygulamalarını test ederken (özellikle şirket içinde geliştirildiyse) kaynak kodu test amacıyla hazır hale getirilmelidir.

Birçok kasıtsız ancak önemli güvenlik sorununun, penetrasyon testi gibi diğer analiz veya test biçimleriyle keşfedilmesi son derece zordur. Bu, kaynak kodu analizini teknik test için tercih edilen teknik hale getirir. Kaynak koduyla, bir test cihazı ne olduğunu (veya olması gereken) doğru bir şekilde belirleyebilir ve kara kutu testinin tahmin çalışmasını kaldırabilir.

Kaynak kodu incelemeleri yoluyla özellikle elverişli olan sorunlara örnek olarak, eşzamanlılık sorunları, kusurlu iş mantığı, erişim kontrol sorunları ve kriptografik zayıflıklar, arka kapılar, Truva atları, Paskalya yumurtaları, zaman bombaları, mantık bombaları ve diğer kötü amaçlı kod formlarını içerir. Bu konular genellikle web uygulamalarındaki en zararlı güvenlik açıkları olarak kendilerini gösterir.

Kaynak kodu analizi, giriş doğrulamasının yapılmadığı veya başarısız check-on kontrol prosedürlerinin mevcut olabileceği yerler gibi uygulama konularını bulmak için de son derece verimli olabilir. Operasyonel prosedürlerin de gözden geçirilmesi gerekir, çünkü dağıtılan kaynak kodu burada analiz edilenle aynı olmayabilir. Ken Thompson'ın Turing Ödülü konuşması bu konunun olası bir tezahürünü anlatıyor.

Advantages (Avantajları)

- Komplelik ve etkinlik
- Doğruluk
- Hızlı (yetkili eleştirmenler için)

Disadvantages (Dezavantajları)

- Yüksek vasıflı güvenlik bilinci geliştiricileri gerektirir
- Derlenmiş kütüphanelerdeki sorunları kaçırabilir
- Çalışma süresi hatalarını kolayca algılayamam
- Aslında dağıtılan kaynak kodu analiz edilenden farklı olabilir

Kod incelemesi hakkında daha fazla bilgi için, OWASP kodu inceleme projesine bakın.

2.7 Penetration Testing (Penetrasyon Testi)

Owerview (Genel Bakış)

Penettrasyon testi, on yıllardır ağ güvenliğini test etmek için kullanılan yaygın bir teknik olmuştur. Ayrıca yaygın olarak kara kutu testi veya etik hackleme olarak da bilinir. Penettrasyon testi, esasen, hedefin kendisinin iç işleyişini bilmeden, güvenlik açıklarını bulmak için bir sistemi veya uygulamayı uzaktan test etmenin "sanatıdır". Tipik olarak, penetrasyon testi ekibi bir uygulamaya kullanıcıymış gibi erişebilir. Testçi bir saldırgan gibi davranır ve güvenlik açıklarını bulmaya ve sömürmeye çalışır. Birçok durumda testere sistemde bir veya daha fazla geçerli hesap verilecektir.

Penettrasyon testinin ağ güvenliğinde etkili olduğu kanıtlanırken, teknik doğal olarak uygulamalara dönüşmez. Ağlarda ve işletim sistemlerinde penetrasyon testi

yapıldığında, ilgili çalışmaların çoğu, belirli teknolojilerde bilinen güvenlik açıklarını bulmakta ve daha sonra kullanmaktır. Web uygulamaları neredeyse sadece ısmarlama olduğundan, web uygulama alanındaki penetrasyon testi saf araştırmaya daha çok benzer. Bazı otomatik penetrasyon test araçları geliştirildi, ancak web uygulamalarının ısmarlama doğası göz önüne alındığında, etkinlikleri tek başına zayıf olabilir.

Birçok kişi birincil güvenlik test tekniği olarak web uygulaması penetrasyon testini kullanır. Kesinlikle bir test programında yerini olsa da, birincil veya tek test tekniği olarak düşünülmesi gerektiğine
inanmıyoruz. Gary McGraw'ın Yazılım Penetrasyon Testi'nde yazdığı gibi, "Uygulamada, bir penetrasyon testi, bir sistemdeki tüm olası güvenlik risklerinin sadece küçük bir temsili örneğini tanımlayabilir." Bununla birlikte, odaklanmış penetrasyon testi (yani, önceki incelemelerde tespit edilen bilinen güvenlik açıklarından yararlanmaya yönelik testler), bazı belirli güvenlik açıklarının fiilen dağıtılan kaynak kodunda düzeltilip sabitlenmediğini tespit etmede yararlı olabilir.

Advantages (Avantajları)

- Hızlı olabilir (ve bu nedenle ucuz)
- Kaynak kodu incelemesinden nispeten daha düşük bir beceri seti gerektirir
- Aslında açığa çıkan kodu test eder

Disadvantages (Dezavantajları)

- SDLC'de çok geç
- Sadece ön darbe testi

2.8 The Need for a Balanced Approach (Dengeli Bir Yaklaşım İhtiyacı)

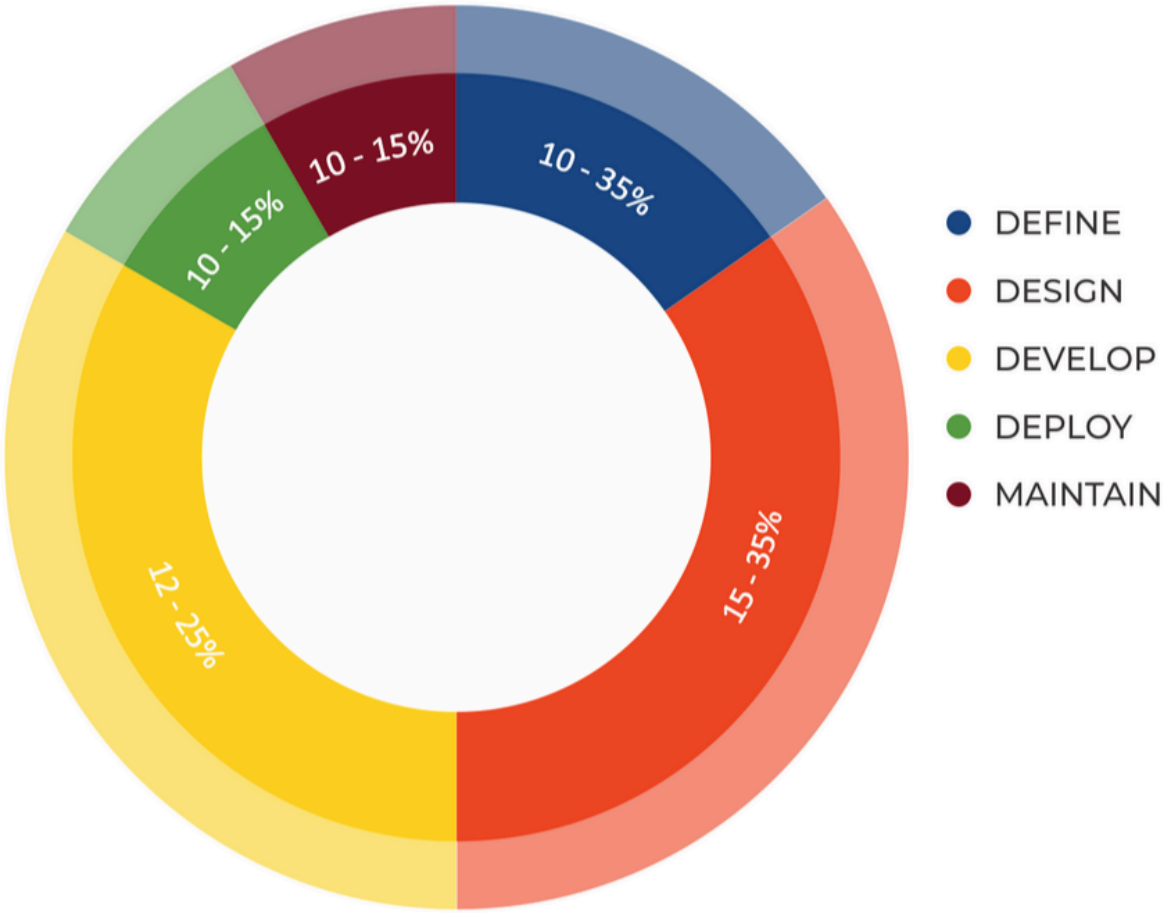
Web uygulamalarının güvenliğini test etmek için bu kadar çok teknik ve yaklaşımla, hangi tekniklerin kullanılacağını veya ne zaman kullanılacağını anlamak zor olabilir. Deneyimler, bir test çerçevesi oluşturmak için tam olarak hangi tekniklerin kullanılması gerektiği sorusuna doğru veya yanlış bir cevap olmadığını göstermektedir. Aslında, tüm teknikler test edilmesi gereken tüm alanları test etmek için kullanılmalıdır.

Tüm güvenlik testlerini etkili bir şekilde karşılamak ve tüm sorunların ele alınmasını sağlamak için gerçekleştirilebilecek tek bir teknik olmadığı açık olsa da, birçok şirket sadece bir yaklaşım benimser. Kullanılan tek yaklaşım tarihsel olarak penetrasyon testi olmuştur. Penettrasyon testi, yararlı olsa da, test edilmesi gereken sorunların çoğunu etkili bir şekilde ele alamaz. SDLC'de sadece "çok geç".

Doğru yaklaşım, manuel incelemelerden teknik teste, CI / CD entegre teste kadar çeşitli teknikleri içeren dengeli bir yaklaşımdır. Dengeli bir yaklaşım, SDLC'nin tüm aşamalarında testleri kapsamalıdır. Bu yaklaşım, mevcut SDLC aşamasına bağlı olarak mevcut en uygun teknikleri kullanır.

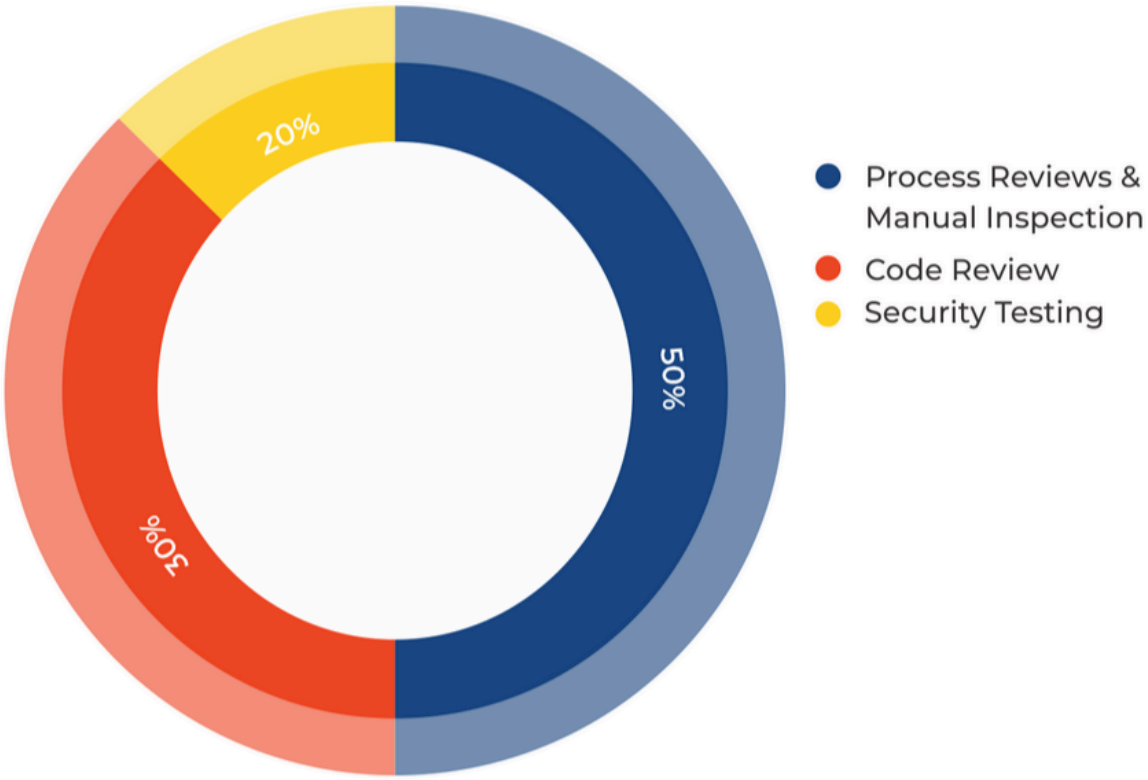
Elbette sadece bir tekniğin mümkün olduğu zamanlar ve koşullar vardır. Örneğin, daha önce oluşturulmuş bir web uygulamasının testini düşünün, ancak test partisinin kaynak koduna erişemediği bir yer. Bu durumda, penetrasyon testi açıkça hiçbir testten daha iyidir. Bununla birlikte, test tarafları, kaynak koduna erişime sahip olmamak ve daha eksiksiz test olasılığını araştırmak gibi varsayımlara meydan okumaya teşvik edilmelidir.

Dengeli bir yaklaşım, test sürecinin olgunluğu ve kurum kültürü gibi birçok faktöre bağlı olarak değişir. Dengeli bir test çerçevesinin Şekil 3 ve Şekil 4'te gösterilen temsiller gibi görünmesi önerilir. Aşağıdaki rakam, SLDC'ye örtülü tipik bir oransal temsil olduğunu göstermektedir. Araştırma ve deneyime uygun olarak, şirketlerin gelişimin ilk aşamalarına daha yüksek bir önem vermesi önemlidir.



Şekil 2-3: SDLC'de Test Çabasının Oranı

Aşağıdaki rakam, test tekniklerine bağlı tipik bir oransal temsili göstermektedir.



Şekil 2-4: Test Tekniğine Göre Test Çabasının Orantı

A Note about Web Application Scanners (Web Uygulama Tarayıcıları Hakkında Bir Not)

Birçok kuruluş otomatik web uygulama tarayıcıları kullanmaya başlamıştır. Kuşkusuz bir test programında yer olsalar da, kara kutu testinin otomatikleştirilmesinin tamamen etkili olmadığına (ne de asla olmayacağına) neden inanıldığı konusunda bazı temel konuların vurgulanması gerekir. Bununla birlikte, bu sorunları vurgulamak, web uygulama tarayıcılarının kullanımını engellememelidir. Aksine, amaç sınırlamaların anlaşılmasını ve test çerçevelerinin uygun şekilde planlanmasını sağlamaktır.

Otomatik güvenlik açığı algılama araçlarının etkinliğini ve sınırlamalarını anlamak yararlıdır. Bu amaçla, OWASP Benchmark Projesi, otomatik yazılım güvenlik açığı algılama araçlarının ve hizmetlerinin hızını, kapsamını ve doğruluğunu değerlendirmek için tasarlanmış bir test süitidir. Kriterleme, bu otomatik araçların yeteneklerini test etmeye yardımcı olabilir ve kullanışlılıklarını açıklığa kavuşturmaya yardımcı olabilir.

Aşağıdaki örnekler, otomatik kara kutu testinin neden etkili olmayabileceğini göstermektedir.

Example 1: Magic Parameters (Örnek 1: Sihirli Parametreler)

Bir isim-değer çifti "sihirli" ve daha sonra değeri kabul eden basit bir web uygulaması hayal edin. Basitlik için, GET talebi şunları olabilir: `http://www.host/application?magic=value`

Örtüsü daha da basitleştirmek için, bu durumdaki değerler yalnızca ASCII karakterleri a - z (üst veya alt kasa) ve bütünleştiriciler 0 - 9 olabilir.

Bu uygulamanın tasarımcıları test sırasında idari bir arka kapı oluşturdular, ancak sıradan gözlemcinin keşfetmesini önlemek için onu gizledi. Değeri sf8sfffsfsdsdierdsgfgfg8d (30 karakter) göndererek, kullanıcı daha sonra giriş ve uygulamanın tam kontrolüne sahip bir idari ekranla sunulacaktır. HTTP isteği şimdi:

`http://www.host/application?magic=sf8g7sfjdsurtsdieerwqredsgnfg8d`

Diğer tüm parametrelerin basit iki ve üç karakterli alanlar olduğu göz önüne alındığında, yaklaşık 28 karakterde kombinasyonları tahmin etmeye başlamak mümkün değildir. Bir web uygulama tarayıcısının, 30 karakterin tüm anahtar alanını zorlaması (veya tahmin etmesi) gerekecektir. Bu 30\^28 permutasyon veya trilyonlarca HTTP isteğine kadar. Bu, dijital bir samanlıkta bir elektron.

Bu örnek Sihirli Parametresi kontrolü için kod aşağıdaki gibi görünebilir:

```
public void doPost( HttpServletRequest request, HttpServletResponse response) {
 String magic = "sf8g7sfjdsurtsdieerwqredsgnfg8d";
 boolean admin = magic.equals( request.getParameter("magic"));
 if (admin) doAdmin( request, response);
 else ... // normal processing
}
```

Koda bakarak, güvenlik açığı pratik olarak potansiyel bir sorun olarak sayfadan atlar.

Example 2: Bad Cryptography (Örnek 2: Kötü Kriptografi)

Kriptografi web uygulamalarında yaygın olarak kullanılmaktadır. Bir geliştiricinin, bir kullanıcıyı A sitesinden otomatik olarak B sitesine imzalamak için basit bir kriptografi algoritması yazmaya karar verdiğini hayal edin. Bilgeliklerinde, geliştirici bir kullanıcı A sitesine giriş yaparsa, daha sonra bir MD5 hash işlevini kullanarak bir anahtar oluşturacaklarına karar verir: `Hash { username : date }`

Bir kullanıcı B sitesine geçtiğinde, sorgu dizesindeki anahtarı HTTP yönlendirmesiyle B sitesine gönderirler. Site B, hash'i bağımsız olarak hesaplar ve istek üzerine aktarılan hash ile karşılaştırır. Eşleşirlerse, B sitesi kullanıcıyı iddia ettikleri kullanıcı olarak işaretler.

Planın açıklandığı gibi, yetersizlikler çözülebilir. Şemayı çözen (veya nasıl çalıştığı veya Bugtraq'tan bilgileri indirdiği) herhangi bir kullanıcı olarak oturum açabilir. Bir inceleme veya kod denetimi gibi manuel muayene, bu güvenlik sorununu hızlı bir şekilde ortaya çıkarırdı. Bir kara kutu web uygulaması tarayıcısı güvenlik açığını ortaya çıkarmaz. Her kullanıcıyla değişen 128 bitlik bir hash görürdü ve hash fonksiyonlarının doğası gereği öngörülebilir bir şekilde değişmedi.

A Note about Static Source Code Review Tools (Statik Kaynak Kod İnceleme Araçları Hakkında Bir Not)

Birçok kuruluş statik kaynak kod tarayıcıları kullanmaya başlamıştır. Kuşkusuz kapsamlı bir test programında yer bulurken, bu yaklaşımın tek başına kullanıldığında neden etkili olmadığıyla ilgili bazı temel konuları vurgulamak gerekir. Statik kaynak kodu analizi tek başına tasarımdaki kusurlar nedeniyle sorunları tanımlayamaz, çünkü kodun inşa edildiği bağlamı anlayamaz. Kaynak kodu analiz araçları, kodlama hataları nedeniyle güvenlik sorunlarının belirlenmesinde yararlıdır, ancak bulguları doğrulamak için önemli manuel çaba gereklidir.

2.9 Deriving Security Test Requirements (Güvenlik Testi Gereksinimlerinin Türetilmesi)

Başarılı bir test programına sahip olmak için, test hedeflerinin ne olduğunu bilmek gerekir. Bu hedefler güvenlik şartları ile belirtilmiştir. Bu bölüm, geçerli standartlar ve düzenlemelerden, olumlu uygulama gerekliliklerinden (uygulamanın ne yapması gerektiğinin belirtilmesi) ve negatif başvuru gerekliliklerinden (uygulamanın ne yapmaması gerektiğinin belirtilmesi) ile güvenlik testi gerekliliklerinin nasıl değiştirileceğini ayrıntılı olarak tartışır. Ayrıca, güvenlik gereksinimlerinin SDLC sırasında güvenlik testlerini nasıl etkili bir şekilde yönlendirdiğini ve güvenlik testi verilerinin yazılım güvenlik risklerini etkili bir şekilde yönetmek için nasıl kullanılabileceğini de tartışmaktadır.

Testing Objectives (Test Hedefleri)

Güvenlik testinin amaçlarından biri, güvenlik kontrollerinin beklendiği gibi çalıştığını doğrulamaktır. Bu sayede belgelenmiştir security requirements Bu, güvenlik kontrolünün işlevselliğini tanımlar. Yüksek düzeyde, bu, verilerin yanı sıra hizmetin gizliliğini, bütünlüğünü ve kullanılabilirliğini kanıtlamak anlamına gelir. Diğer amaç, güvenlik kontrollerinin az veya hiç güvenlik açığı olmadan uygulandığını doğrulamaktır. Bunlar, OWASP Top Ten gibi yaygın güvenlik açıklarının yanı sıra daha önce SDLC sırasında tehdit modellemesi, kaynak kodu analizi ve penetrasyon testi gibi güvenlik
değerlendirmeleriyle tanımlanan güvenlik açıklarıdır.

Security Requirements Documentation (Güvenlik Gereksinimleri Dokümanı)

Güvenlik gereksinimlerinin dokümantasyonunda ilk adım, anlamayı sağlamaktır. business requirements . . Bir iş gereksinimi belgesi, uygulamanın beklenen işlevselliği hakkında ilk üst düzey bilgileri sağlayabilir. Örneğin, bir uygulamanın temel amacı müşterilere finansal hizmetler sunmak veya ürünlerin çevrimiçi bir katalogdan satın alınmasına izin vermek olabilir. İş gereksinimlerinin bir güvenlik bölümü, müşteri verilerini korumanın yanı sıra yönetmelikler, standartlar ve politikalar gibi geçerli güvenlik belgelerine uyma ihtiyacını vurgulamalıdır.

Geçerli düzenlemelerin, standartların ve politikaların genel bir kontrol listesi, web uygulamaları için iyi bir ön güvenlik uyumluluk analizidir. Örneğin, uyum düzenlemeleri, iş sektörü ve başvurunun
işleyeceği ülke veya devlet hakkında bilgi kontrol edilerek tanımlanabilir. Bu uyumluluk yönergeleri ve düzenlemelerinin bazıları, güvenlik kontrolleri için belirli teknik gerekliliklere dönüşebilir.
Örneğin, finansal uygulamalar söz konusu olduğunda, Federal Finansal Kurumlar Sınav Konseyi (FFIEC) Siber Güvenlik Değerlendirme Aracı ve Dokümantı'na uyum, finansal kurumların çok katmanlı güvenlik kontrolleri ve çok faktörlü kimlik doğrulama ile zayıf kimlik doğrulama risklerini
hafifleten uygulamaları uygulamasını gerektirir.

Güvenlik için uygulanabilir endüstri standartları da genel güvenlik gereksinimi kontrol listesi tarafından ele geçirilmelidir. Örneğin, müşteri kredi kartı verilerini işleyen uygulamalar söz konusu olduğunda, PCI Güvenlik Standartları Konseyi Veri Güvenliği Standardına (DSS) uygunluk, PIN'lerin ve CVV2 verilerinin depolanmasını yasaklar ve tüccarın manyetik şerit verilerin şifreleme ile ve iletimle maskeleme ile

korumasını gerektirir. Bu PCI DSS güvenlik gereksinimleri kaynak kodu analizi ile doğrulanabilir.

Kontrol listesinin bir başka bölümünün, kuruluşun bilgi güvenliği standartlarına ve politikalarına uygunluk için genel gereklilikleri uygulaması gerekir. İşlevsel gereksinimler perspektifinden bakıldığında, güvenlik kontrolü için gerekliliklerin bilgi güvenliği standartlarının belirli bir bölümüne eşleştirilmesi gerekir. Böyle bir gereksinimin bir örneği şunlar olabilir: "On alfanümerik karakterin şifre karmaşıklığı, uygulama tarafından kullanılan kimlik doğrulama kontrolleri tarafından uygulanmalıdır." Güvenlik gereklilikleri uyumluluk kurallarına göre haritalandığında, bir güvenlik testi uyumluluk risklerinin maruziyetini doğrulayabilir. Bilgi güvenliği standartları ve politikaları ile ihlal bulunursa, bunlar belgelenebilecek ve işletmenin yönetmesi veya ele alması gereken bir riskle sonuçlanacaktır. Bu güvenlik uyumu gereklilikleri uygulanabilir olduğundan, iyi belgelenmiş ve güvenlik testleriyle doğrulanmaları gerekir.

Security Requirements Validation (Güvenlik Şartları Doğrulama)

İşlevsellik perspektifinden bakıldığında, güvenlik gereksinimlerinin doğrulanması güvenlik testinin temel amacıdır. Risk yönetimi perspektifinden bakıldığında, güvenlik gereksinimlerinin doğrulanması bilgi güvenliği değerlendirmelerinin hedefidir. Yüksek düzeyde, bilgi güvenliği değerlendirmelerinin ana hedefi, temel kimlik doğrulama, yetkilendirme veya şifreleme kontrolleri gibi güvenlik kontrollerindeki boşlukların tanımlanmasıdır. Daha fazla incelendiğinde, güvenlik değerlendirmesi hedefi, verilerin gizliliğini, bütünlüğünü ve kullanılabilirliğini sağlayan güvenlik kontrollerindeki potansiyel zayıflıkların tanımlanması gibi risk analizidir. Örneğin, uygulama kişisel olarak tanımlanabilir bilgiler (PII) ve hassas verilerle uğraştığında, doğrulanması gereken güvenlik gereksinimi, bu tür verilerin transit ve depolamada şifrelenmesini gerektiren şirket bilgi güvenliği politikasına uygunluktur. Şifrelemenin verilerin, şifreleme algoritmalarını ve temel uzunlukları korumak için kullanıldığını varsayarsak, kuruluşun şifreleme standartlarına uyması gerekir. Bunlar sadece belirli algoritmaların ve anahtar uzunlukların kullanılmasını gerektirebilir. Örneğin, test edilebilir bir güvenlik gereksinimi, yalnızca izin verilen şifrelerin (örneğin, SHA-256, RSA, AES) izin verilen minimum anahtar uzunlukları (örneğin, simetrik için 128 bit ve asimetrik şifreleme için 1024'ten fazla) kullanıldığını doğrulamaktır.

Güvenlik değerlendirme perspektifinden bakıldığında, güvenlik gereksinimleri farklı eserler ve test metodolojileri kullanarak SDLC'nin farklı aşamalarında doğrulanabilir. Örneğin, tehdit modellemesi tasarım sırasında güvenlik kusurlarını belirlemeye odaklanır; güvenli kod analizi ve incelemeleri, geliştirme sırasında kaynak kodundaki güvenlik sorunlarını belirlemeye odaklanır; ve penetrasyon testi, test veya doğrulama sırasında uygulamadaki güvenlik açıklarını belirlemeye odaklanır.

SDLC'nin başlarında tespit edilen güvenlik sorunları bir test planında belgelenebilir, böylece daha sonra güvenlik testleri ile doğrulanabilir. Farklı test tekniklerinin sonuçlarını birleştirerek daha iyi güvenlik testi vakaları elde etmek ve güvenlik gereksinimlerinin güvence seviyesini artırmak mümkündür. Örneğin, penetrasyon testleri ve kaynak kodu analizinin sonuçları birleştiğinde gerçek güvenlik açıklarını kullanılamaz olanlardan ayırt etmek mümkündür. Örneğin, bir SQL enjeksiyonu güvenlik açığı için güvenlik testi göz önüne alındığında, bir kara kutu testi önce güvenlik açığının parmak izini için uygulamanın bir taramasını içerebilir. Doğrulanabilen potansiyel bir SQL enjeksiyonu güvenlik açığının ilk kanıtı, bir SQL istisnasının üretilmesidir. SQL güvenlik açığının daha fazla doğrulanması, bir bilgi ifşası istismarı için SQL sorguunun dilbilgisini değiştirmek için saldırı vektörlerine manuel olarak enjekte etmeyi içerebilir. Bu, kötü niyetli sorgular uygulanmadan önce çok fazla deneme ve hata analizi içerebilir. Test cihazının kaynak koduna sahip olduğunu varsayarsak, güvenlik açığından başarıyla yararlanacak SQL saldırı vektörünün nasıl oluşturulacağını doğrudan kaynak kodu analizinden öğrenebilirler (örneğin, gizli verileri yetkisiz kullanıcıya iade eden kötü amaçlı bir sorgu yürütün). Bu, SQL güvenlik açığının doğrulanmasını hızlandırabilir.

Threats and Countermeasures Taxonomies (Tehditler ve Karşı Önlemler Taksonomi)

A. threat and countermeasure classification Güvenlik açıklarının kök nedenlerini göz önünde bulunduran , güvenlik kontrollerinin bu tür güvenlik açıklarının maruziyetinin etkisini azaltmak için tasarlanmış, kodlandığını ve inşa edildiğini doğrulamada kritik faktördür. Web uygulamaları söz konusu olduğunda, güvenlik kontrollerinin OWASP Top Ten gibi yaygın güvenlik açıklarına maruz kalması, genel güvenlik gereksinimlerini elde etmek için iyi bir başlangıç noktası olabilir. OWASP Test Kılavuzu Kontrol Listesi, test cihazları belirli güvenlik açıkları ve doğrulama testleri ile rehberlik etmek için yararlı bir kaynaktır.

Bir tehdit ve karşı önlem kategorisinin odak noktası, güvenlik gereksinimlerini tehditler ve kırılganlığın temel nedeni açısından tanımlamaktır. Bir tehdit, Svoofing, Kurcalama, İrtibat, Bilgi
açıklaması, hizmetin inkarı ve ayrıcalıklılığın Elevmesi için bir kısaltma olan STTRIDE kullanılarak kategorize edilebilir. Temel neden, tasarımdaki güvenlik kusuru, kodlamada bir güvenlik hatası veya güvensiz konfigürasyon nedeniyle bir sorun olarak kategorize edilebilir. Örneğin, zayıf kimlik
doğrulama zammarılığın temel nedeni, veriler uygulamanın istemcisi ve sunucu katmanları arasındaki bir güven sınırını geçtiğinde karşılıklı kimlik doğrulama eksikliği olabilir. Bir mimari tasarım incelemesi sırasında itibarsızlaştırma tehdidini yakalayan bir güvenlik gereksinimi, daha sonra güvenlik testleri ile doğrulanabilen karşı önlem (örneğin, karşılıklı kimlik doğrulama) gerekliliğinin belgelenmesine izin verir.

Güvenlik açıkları için bir tehdit ve karşı önlem sınıflandırması, güvenli kodlama standartları gibi güvenli kodlama için güvenlik gereksinimlerini belgelemek için de kullanılabilir. Kimlik doğrulama kontrollerinde ortak bir kodlama hatası örneği, bir tohumu değere uygulamadan bir şifreyi şifrelemek için bir hash fonksiyonu uygulamaktır. Güvenli kodlama perspektifinden bakıldığında, bu, bir kodlama hatasında bir güvenlik açığı kökü nedeni ile kimlik doğrulama için kullanılan şifrelemeyi etkileyen bir güvenlik açığıdır. Kök neden güvensiz kodlama olduğundan, güvenlik gereksinimi güvenli kodlama standartlarında belgelenebilir ve SDLC'nin geliştirme aşamasında güvenli kod incelemeleri yoluyla doğrulanabilir.

Security Testing and Risk Analysis (Güvenlik Test ve Risk Analizi)

Güvenlik gereklilikleri, bir desteği desteklemek için güvenlik açıklarının ciddiyetini dikkate alması gerekir. risk mitigation strategy . . Kuruluşun uygulamalarda (yani bir güvenlik açığı bilgi tabanında) bulunan güvenlik açıklarının deposunu koruduğunu varsayarsak, güvenlik sorunları tür, sorun, hafifletme, kök nedene göre rapor edilebilir ve bulundukları uygulamalara göre rapor edilebilir. Böyle bir güvenlik açığı bilgi tabanı, SDLC genelindeki güvenlik testlerinin etkinliğini
analiz etmek için bir metrik oluşturmak için de kullanılabilir.

Örneğin, kaynak kodu analizi yoluyla tanımlanan ve kodlama hatası kökü nedeni ve giriş doğrulama güvenlik açığı türü ile bildirilen SQL enjeksiyonu gibi bir giriş doğrulama sorununu düşünün. Bu tür güvenlik açığının maruz kalması, birkaç SQL enjeksiyon saldırısı vektörü ile giriş alanlarını araştırarak bir penetrasyon testi ile

değerlendirilebilir. Bu test, veritabanına çarpmadan ve güvenlik açığını hafifletmeden önce özel karakterlerin filtrelendiğini doğrulayabilir. Kaynak kod analizi ve penetrasyon testinin sonuçlarını birleştirerek, güvenlik açığının olasılığını ve maruziyetini belirlemek ve güvenlik açığının risk derecesini hesaplamak mümkündür. Bulgularda güvenlik açığı risk derecelendirmelerini bildirerek (örneğin, test raporu) hafifletme stratejisine karar vermek mümkündür. Örneğin, yüksek ve orta riskli güvenlik açıkları iyileştirme için önceliklendirilebilirken, düşük riskli güvenlik açıkları gelecekteki sürümlerde sabitlenebilir.

Ortak güvenlik açıklarından yararlanma tehdidi senaryolarını göz önünde bulundurarak, uygulama güvenlik kontrolünün test edilmesi gereken potansiyel riskleri belirlemek mümkündür. Örneğin, OWASP Top Ten güvenlik açıkları, kimlik avı, gizlilik ihlalleri, tanımlama, hırsızlık, sistem uzlaşması, veri değişikliği veya veri tahribatı, finansal kayıp ve itibar kaybı gibi saldırılarla eşleştirilebilir. Bu tür konular tehdit senaryolarının bir parçası olarak belgelenmelidir. Tehdit ve güvenlik açıkları açısından düşünerek, bu tür saldırı senaryolarını simüle eden bir test bataryası tasarlamak mümkündür. İdeal olarak, kuruluşun güvenlik açığı bilgi tabanı, en olası saldırı senaryolarını doğrulamak için güvenlik riski odaklı test vakaları türetmek için kullanılabilir. Örneğin, kimlik hırsızlığı yüksek riskli olarak kabul edilirse, negatif test senaryoları kimlik doğrulama, kriptografik kontroller, giriş doğrulaması ve yetkilendirme kontrollerinde güvenlik açıklarından elde edilen etkilerin hafifletilmesini doğrulamalıdır.

Deriving Functional and Non-Functional Test Requirements (Fonksiyonel ve Fonksiyonel Olmayan Test Gereksinimleri)

Functional Security Requirements (Fonksiyonel Güvenlik Gereksinimleri)

İşlevsel güvenlik gereksinimleri perspektifinden, geçerli standartlar, politikalar ve düzenlemeler hem bir tür güvenlik kontrolüne hem de kontrol işlevselliğine olan ihtiyacı artırır. Bu gereksinimler aynı zamanda “pozitif gereksinimler” olarak da adlandırılır, çünkü güvenlik testleri yoluyla doğrulanabilecek beklenen işlevselliği belirtirler. Olumlu gereksinimlere örnek olarak: “Uygulama, altı başarısız girişimden sonra kullanıcıyı kilitleyecektir” veya “şiddet kelimelerinin en az on alfanümerik karakter olması gerekir”. Olumlu gerekliliklerin doğrulanması, beklenen işlevselliği iddia etmekten oluşur ve test koşullarını yeniden oluşturarak ve testi önceden tanımlanmış girdilere göre çalıştırarak test edilebilir. Sonuçlar daha sonra bir başarısızlık veya geçiş koşulu olarak gösterilir.

Güvenlik testlerinde güvenlik gerekliliklerini doğrulamak için güvenlik gereksinimlerinin işlev odaklı olması gerekir. Beklenen işlevselliği (ne) vurgulamaları ve uygulamayı (nasıl) ima etmeleri gerekir. Kimlik doğrulama için üst düzey güvenlik tasarım gereksinimlerine ilişkin örnekler şunları olabilir:

- Kullanıcı kimlik bilgilerini veya paylaşılan sırları transit ve depolamada koruyun.
- Ekrandaki gizli verileri maskeleyin (örneğin, şifreler, hesaplar).
- Girişimlerde belirli sayıda başarısız kayıttan sonra kullanıcı hesabını kilitleyin.
- Başarısız bir günlük sonucunda kullanıcıya özel doğrulama hataları göstermeyin.
- Sadece alfanümerik olan, özel karakterler içeren ve saldırı yüzeyini sınırlamak için en az on karakter olan şifrelere izin verin.
- Şifre değişikliğinin, şifre değişikliği yoluyla bir şifrenin kaba kuvvetlendirilmesini önlemek için, eski şifreyi, yeni şifreyi vekullanıcının meydan okuma sorusuna verdiği cevabı doğrulayarak yalnızca doğrulanmış kullanıcılara izin verin.
- Şifre sıfırlama formu, geçici şifreyi e-posta yoluyla kullanıcıya göndermeden önce kullanıcının kullanıcı adını ve kullanıcının kayıtlı e-postasını doğrulamalıdır. Verilen geçici şifre tek seferlik bir şifre olmalıdır. Şifre sıfırlama web sayfasına bir bağlantı kullanıcıya gönderilecektir. Şifre sıfırlama web sayfası, kullanıcının geçici şifresini, yeni şifresini ve kullanıcının meydan okuma sorusuna verdiği cevabı doğrulamalıdır.

Risk-Driven Security Requirements (Riskten Kaynaklanan Güvenlik Gereksinimleri)

Güvenlik testleri de risk odaklı olmalıdır. Beklenmedik davranış veya olumsuz gereksinimler için başvuruyu doğrulamaları gerekir.

Negatif gereksinimlere örnek olarak:

- Uygulama, verilerin değiştirilmesine veya yok edilmesine izin vermemelidir.
- Uygulama, kötü niyetli bir kullanıcı tarafından yetkisiz finansal işlemler için tehlikeye atılmamalı veya kötüye kullanılmamalıdır.

Olumsuz gereklilikleri test etmek daha zordur, çünkü aranması beklenen bir davranış yoktur. Yukarıdaki gereksinimlere uyacak şekilde beklenen davranışın aranması, öngörülemeyen girdi koşulları, nedenler ve etkilerle gerçekçi olmayan bir tehdit analistinin ortaya çıkmasını gerektirebilir. Bu nedenle, güvenlik testinin risk analizi ve tehdit modellemesi tarafından yönlendirilmesi gerekir. Anahtar, tehdit senaryolarını ve karşı önlemin işlevselliğini bir tehdidi hafifletmek için bir faktör olarak belgelemektir.

Örneğin, kimlik doğrulama kontrolleri söz konusu olduğunda, aşağıdaki güvenlik gereksinimleri tehditlerden ve karşı önlemler perspektifinden belgelenebilir:

- Bilgi açıklama ve kimlik doğrulama protokolü saldırıları riskiniazaltmak için depolama ve transitte kimlik doğrulama verilerini şifreleyin.
- Sözlük saldırılarını önlemek için bir sindirme (örneğin HASH) ve bir tohum kullanmak gibi geri dönüşümsüz şifrelemeyi kullanarak şifreleri şifreleyin.
- Başarısızlık eşiğinde bir günlüğe ulaştıktan sonra hesapları kapatın ve kaba kuvvet şifresi saldırı riskini azaltmak için şifre karmaşıklığını uygulayın.
- Hesap toplama veya numaralandırma riskini azaltmak için kimlik bilgilerinin doğrulanması üzerine jenerik hata mesajları görüntüleyin.
- İtfa edilmemesi ve Ortadaki (MiTM) saldırılarını önlemek için istemci ve sunucuyu karşılıklı olarak doğrulayın.

Tehdit ağaçları ve saldırı kütüphaneleri gibi tehdit modelleme araçları negatif test senaryolarını türetmeye yararlı olabilir. Bir tehdit ağacı, bir kök saldırıyı (örneğin, saldırgan diğer kullanıcıların mesajlarını okuyabilecek) ve güvenlik kontrollerinin farklı istismarlarını (örneğin, bir SQL enjeksiyonu güvenlik açığı nedeniyle veri doğrulaması başarısız olur) ve bu tür saldırıları hafifletmede etkili olmak üzere doğrulanabilecek gerekli karşı önlemleri (örneğin, veri doğrulamasını ve parametrilenmiş sorguları) tanımlayacaktır.

Deriving Security Test Requirements Through Use and Misuse Cases (Kullanım ve Yanlış Kullanım Durumları Yoluyla Güvenlik Testi Gereksinimlerinin Verilmesi)

Uygulama işlevselliğini tanımlamanın bir ön koşulu, uygulamanın ne yapması ve nasıl yapılacağını anlamaktır. Bu, kullanım davalarını tarif ederek yapılabilir. Kullanım, yazılım mühendisliğinde yaygın olarak kullanıldığı gibi grafik formunda, aktörlerin etkileşimlerini ve ilişkilerini gösterir. Uygulamadaki aktörleri, ilişkileri, her

senaryo için amaçlanan eylem sırası, alternatif eylemler, özel gereksinimler, önkoşullar ve koşullar sonrası tanımlamaya yardımcı olurlar.

Kullanıma benzer şekilde, yanlış kullanım veya kötüye kullanım davaları, uygulamanın istenmeyen ve kötü amaçlı kullanım senaryolarını tanımlar. Bu yanlış kullanım durumları, bir saldırganın uygulamayı nasıl kötüye kullanabileceğine ve kötüye kullanabileceğine dair senaryoları tanımlamanın bir yolunu sunar. Bir kullanım senaryosunda bireysel adımlardan geçerek ve nasıl kötü niyetli olarak kullanılabileceğini düşünerek, uygulamanın iyi tanımlanmayan potansiyel kusurları veya yönleri keşfedilebilir. Anahtar, mümkün olan her şeyi veya en azından en kritik kullanım ve yanlış kullanım senaryolarını tanımlamaktır.

Yanlış kullanım senaryoları, uygulamanın saldırganın bakış açısından analizine izin verir ve potansiyel güvenlik açıklarına maruz kalmanın neden olduğu etkiyi azaltmak için uygulanması gereken potansiyel güvenlik açıklarını ve karşı önlemlerin belirlenmesine katkıda bulunur. Tüm kullanım ve suistimal vakaları göz önüne alındığında, hangilerinin en kritik olduğunu ve güvenlik gereksinimlerinde belgelenmesi gerektiğini belirlemek için bunları analiz etmek önemlidir. En kritik suistimal ve suistimal davalarının tanımlanması, güvenlik gereksinimlerinin belgelendirilmesini ve güvenlik risklerinin hafifletilmesi gereken gerekli kontrolleri yönlendirir.

Hem kullanım hem de kötüye kullanım durumlarından güvenlik gereksinimlerini elde etmek için,
işlevsel senaryoları ve olumsuz senaryoları tanımlamak ve bunları grafiksel biçimde koymak önemlidir. Aşağıdaki örnek, kimlik doğrulama için güvenlik gereksinimlerinin türetilmesi durumu için adım adım bir metodolojidir.

Step 1: Describe the Functional Scenario (Adım 1: Fonksiyonel Senaryoyu Tarif Edin)

Kullanıcı kullanıcı adı ve şifre sağlayarak doğrular. Uygulama, kullanıcı kimlik bilgilerinin uygulama tarafından kimlik doğrulamasına dayanarak kullanıcılara erişim sağlar ve doğrulama başarısız olduğunda kullanıcıya özel hatalar sağlar.

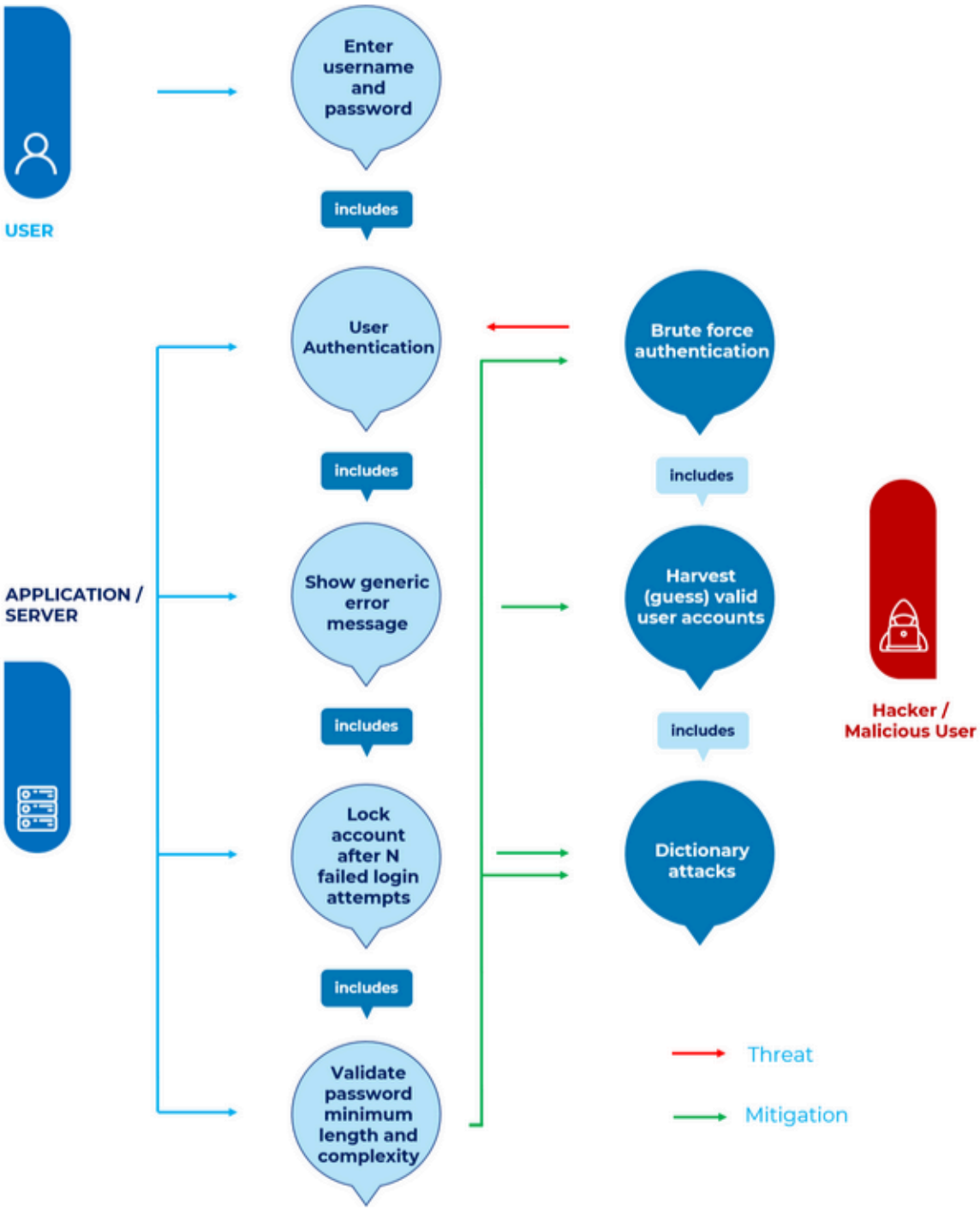
Step 2: Describe the Negative Scenario (Adım 2: Negatif Senaryoyu Tarif Edin)

Saldırgan, uygulamadaki şifrelerin ve hesap hasat açıklarının kaba bir kuvveti veya sözlük saldırısı yoluyla kimlik doğrulamasını bozar. Doğrulama hataları, hangi hesapların geçerli kayıtlı hesaplar (kullanıcı adlar) olduğunu tahmin etmek için

kullanılan bir saldırgana özel bilgiler sağlar. Saldırgan daha sonra geçerli bir hesap için şifreyi zorlamaya çalışır. Şifrelere minimum dört haneli ağır bir kuvvet saldırısı, sınırlı sayıda girişimle (yani, 10\^4) ile başarılı olabilir.

Step 3: Describe Functional and Negative Scenarios with Use and Misuse Case (Adım 3: Fonksiyonel ve Negatif Senaryoları Kullanım ve Yanlış Kullanım Davası ile Tanımlayın)

Aşağıdaki grafiksel örnek, güvenlik gereksinimlerinin kullanım ve kötüye kullanım durumlarıyla türetilmesini göstermektedir. İşlevsel senaryo, kullanıcı eylemlerinden (bir kullanıcı adı ve şifreye girmek) ve uygulama eylemlerinden oluşur (kullanıcıyı doğrulamak ve doğrulama başarısız olursa bir hata mesajı sağlamak). Yanlış kullanım durumu saldırgan eylemlerinden oluşur, yani bir sözlük saldırısı yoluyla şifreyi zorlayarak ve hata mesajlarından geçerli kullanıcı adlarını tahmin ederek kabarıklıkla kimlik doğrulamasını kırmaya çalışır. Kullanıcı eylemlerine yönelik tehditleri (yanlış kullanımlar) grafiksel olarak temsil ederek, karşı önlemleri, bu tür tehditleri hafifleten uygulama eylemleri olarak türetilebilir.



Şekil 2-5: Kullanım ve Yanlış Kullanım Kılıfı

Step 4: Elicit the Security Requirements (Adım 4: Güvenlik Gereksinimlerini Ortaya Çıkarın)

Bu durumda, kimlik doğrulama için aşağıdaki güvenlik gereksinimleri türetilmiştir:

1. Şifre gereksinimleri, yeterli karmaşıklık için mevcut standartlar ile uyumlu olmalıdır.
2. Hesaplar, girişimlerde beş başarısız kayıttan sonra kilitlenmek olmalıdır.
3. Hatalı mesajlarda giriş jenerik olmalıdır.

Bu güvenlik gereksinimleri belgelenmesi ve test edilmesi gerekir.

2.10 Security Tests Integrated in Development and Testing Workflows (Geliştirme ve Test İş Akışlarına Entegre Edilmiş Güvenlik Testleri)

Security Testing in the Development Workflow (Geliştirme İş akışında güvenlik testi)

SDLC'nin geliştirme aşamasındaki güvenlik testi, geliştiricilerin geliştirdikleri bireysel yazılım bileşenlerinin diğer bileşenlerle entegre edilmeden veya uygulamaya yerleştirilmeden önce güvenlik test edilmesini sağlamaları için ilk fırsatı temsil eder. Yazılım bileşenleri, işlevler, yöntemler ve sınıflar gibi yazılım eserlerinin yanı sıra uygulama programlama arayüzleri, kütüphaneler ve yürütülebilir dosyalar gibi yazılım eserlerinden oluşabilir. Güvenlik testi için geliştiriciler, gelişmiş kaynak kodunun potansiyel güvenlik açıklarını içermediğini ve güvenli kodlama standartlarına uygun olduğunu statik olarak doğrulamak için kaynak kod analizinin sonuçlarına güvenebilirler. Güvenlik birimi testleri, bileşenlerin beklendiği gibi çalıştığı dinamik olarak (yani, çalışma zamanında) daha fazla doğrulayabilir. Uygulama yapısında hem yeni hem de mevcut kod değişikliklerini entegre etmeden önce, statik ve dinamik analiz sonuçları gözden geçirilmeli ve doğrulanmalıdır.

Uygulama yapılarında entegrasyondan önce kaynak kodun doğrulanması genellikle kıdemli geliştiricinin sorumluluğundadır. Kıdemli geliştiriciler genellikle yazılım güvenliği konusunda konu uzmanlarıdır ve rolleri güvenli kod incelemesine liderlik etmektir. Uygulama yapımında yayınlanacak kodu kabul edip etmeme veya daha fazla değişiklik ve test gerektirmeye karar vermeleri gerekir. Bu güvenli kod inceleme iş akışı, resmi kabulün yanı sıra bir iş akışı yönetim aracında bir kontrol yoluyla uygulanabilir. Örneğin, fonksiyonel hatalar için kullanılan tipik kusur yönetimi iş akışını varsayarsak, bir geliştirici tarafından düzeltilmiş güvenlik hataları veya değişim yönetim sistemi hakkında rapor edilebilir. Yapı ustası daha sonra,

araçtaki geliştiriciler tarafından bildirilen test sonuçlarına bakabilir ve uygulama yapısına kod değişikliklerini kontrol etmek için onaylar verebilir.

Security Testing in the Test Workflow (Test İş akışında güvenlik testi)

Bileşenler ve kod değişiklikleri geliştiriciler tarafından test edildikten ve uygulama oluşturmasına kontrol edildikten sonra, yazılım geliştirme süreci iş akışındaki en olası bir sonraki adım, uygulamada tüm bir varlık olarak testler yapmaktır. Bu test seviyesi genellikle entegre test ve sistem seviyesi testi olarak adlandırılır. Güvenlik testleri bu test faaliyetlerinin bir parçası olduğunda, hem bir bütün olarak uygulamanın güvenlik işlevselliğini hem de başvuru seviyesi güvenlik açıklarına maruz kalmayı doğrulamak için kullanılabilirler. Uygulamadaki bu güvenlik testleri, kaynak kodu analizi gibi hem beyaz kutu testini hem de penetrasyon testi gibi kara kutu testini içerir. Testler ayrıca, testçinin uygulama hakkında kısmi bilgiye sahip olduğu varsayıldığı gri kutu testi de içerebilir. Örneğin, uygulamanın oturum yönetimi hakkında bazı bilgilerle, test cihazı giriş ve zaman aşımı fonksiyonlarının düzgün bir şekilde güvence altına alınıp alınmadığını daha iyi anlayabilir.

Güvenlik testlerinin hedefi, saldırıya karşı savunmasız olan tam sistemdir. Bu aşamada güvenlik testçilerinin güvenlik açıklarının sömürülüp kullanılabileceğini belirlemesi mümkündür. Bunlar, ortak web uygulama güvenlik açıklarının yanı sıra, daha önce SDLC'de tehdit modellemesi, kaynak kodu analizi ve güvenli kod incelemeleri gibi diğer faaliyetlerle tanımlanan güvenlik sorunlarını içerir.

Genellikle, mühendisleri test etmek, daha ziyade yazılım geliştiricileri, uygulama entegrasyon sistemi testleri için kapsam olarak kullanıldığında güvenlik testleri yapar. Test mühendisleri, web uygulama güvenlik açıkları, kara kutu ve beyaz kutu test teknikleri hakkında güvenlik bilgisine sahiptir ve bu aşamada güvenlik gereksinimlerinin doğrulanmasına sahiptir. Güvenlik testleri yapmak için güvenlik testi vakalarının güvenlik test yönergelerinde ve prosedürlerinde belgelenmesi bir ön koşuldur.

Uygulamanın güvenliğini entegre sistem ortamında doğrulayan bir test mühendisi, operasyonel ortamda test için uygulamayı (örneğin, kullanıcı kabul testleri) yayınlayabilir. SDLC'nin bu aşamasında (yani doğrulama), uygulamanın işlevsel testi genellikle QA test cihazlarının sorumluluğundayken, beyaz şapkalı bilgisayar korsanları veya güvenlik danışmanları genellikle güvenlik testlerinden sorumludur. Bazı kuruluşlar, üçüncü taraf değerlendirmesi gerekli olmadığında (denetleme

amaçlı) bu tür testleri yapmak için kendi özel etik bilgisayar korsanlığı ekibine güvenirler.

Bu testler bazen başvuru üretime açıklanmadan önce güvenlik açıklarını gidermek için son savunma hattı olabileceğinden, sorunların test ekibi tarafından önerildiği gibi ele alınması önemlidir. Öneriler kod, tasarım veya yapılandırma değişikliği içerebilir. Bu seviyede, güvenlik denetçileri ve bilgi güvenliği görevlileri bildirilen güvenlik konularını tartışır ve bilgi risk yönetimi prosedürlerine göre potansiyel riskleri analiz eder. Bu tür prosedürler, geliştirme ekibinin, bu tür riskler kabul edilmedikçe ve kabul edilmedikçe, uygulama konuşlandırılmadan önce tüm yüksek riskli güvenlik açıklarını düzeltmesini gerektirebilir.

Developer’s Security Tests (Geliştiricinin Güvenlik Testleri)

Security Testing in the Coding Phase: Unit Tests (Kodlama Aşamasında Güvenlik Testi: Birim Testleri)

Geliştiricinin bakış açısına göre, güvenlik testlerinin temel amacı, kodun güvenli kodlama standartları gereksinimlerine uygun olarak geliştirildiğini doğrulamaktır. Geliştiricilerin kendi kodlama eserleri (işlevler, yöntemler, sınıflar, API'ler ve kütüphaneler gibi) uygulama yapısına entegre edilmeden önce işlevsel olarak doğrulanmalıdır.

Geliştiricilerin takip etmesi gereken güvenlik gereksinimleri, güvenli kodlama standartlarında belgelenmeli ve statik ve dinamik analiz ile doğrulanmalıdır. Birim test etkinliği güvenli bir kod incelemesini takip ederse, birim testler güvenli kod incelemelerinin gerektirdiği kod değişikliklerinin uygun şekilde uygulandığını doğrulayabilir. Kaynak kod analizi araçları aracılığıyla hem güvenli kod incelemeleri hem de kaynak kodu analizi, geliştiricilerin kaynak kodundaki güvenlik sorunlarını geliştirildiği gibi tanımlamalarına yardımcı olabilir. Birim testleri ve dinamik analiz (örneğin, hata ayıklama) kullanarak geliştiriciler, bileşenlerin güvenlik işlevselliğini doğrulayabilir ve geliştirilen karşı önlemlerin daha önce tehdit modelleme ve kaynak kodu analizi yoluyla tanımlanan güvenlik risklerini hafiflettiğini doğrulayabilir.

Geliştiriciler için iyi bir uygulama, mevcut birim test çerçevesinin bir parçası olan genel bir güvenlik test süiti olarak güvenlik testi kılıfları oluşturmaktır. Genel bir güvenlik test süiti, daha önce tanımlanmış kullanım ve kötüye kullanım vakalarından güvenlik testi işlevlerine, yöntemlerine ve sınıflarına türetilebilir. Genel

bir güvenlik test süiti, aşağıdaki gibi güvenlik kontrolleri için hem pozitif hem de negatif gereksinimleri doğrulamak için güvenlik testi vakalarını içerebilir:

- Kimlik, kimlik doğrulama ve erişim kontrolü
- Giriş doğrulama ve kodlama
- Şifreleme
- Kullanıcı ve oturum yönetimi
- Hata ve istisna işleme
- Denetim ve kütükleme

Geliştiriciler, IDE'lerine entegre edilmiş bir kaynak kodu analiz aracı, güvenli kodlama standartları ve bir güvenlik birimi test çerçevesi ile güçlendirilen yazılım bileşenlerinin güvenliğini değerlendirebilir ve doğrulayabilir. Güvenlik testi vakaları, kaynak kodunda kök nedenleri olan potansiyel güvenlik sorunlarını belirlemek için çalıştırılabilir: bileşenleri giren ve çıkan parametrelerin giriş ve çıktı doğrulamasının yanı sıra, bu konular arasında bileşen tarafından yapılan kimlik doğrulama ve yetkilendirme kontrolleri, bileşen dahilinde verilerin korunması, güvenli istisna ve hata işleme ve güvenli denetim ve oturum açma bulunur. JUnit, NUnit ve CUir gibi birim test çerçeveleri, güvenlik testi gereksinimlerini doğrulamak için uyarlanabilir. Güvenlik fonksiyonel testleri söz konusu olduğunda, birim seviye testleri, güvenlik kontrollerinin işlevlerini işlevlerini, işlevleri, yöntemler veya sınıflar gibi yazılım bileşeni düzeyinde test edebilir. Örneğin, bir test kılıfı, bileşenin beklenen işlevselliğini öne sürerek giriş ve çıkış doğrulamasını (örneğin, değişken sanitasyon) ve değişkenler için sınır kontrollerini doğrulayabilir.

Kullanım ve yanlış kullanım durumlarıyla tanımlanan tehdit senaryoları, yazılım bileşenlerini test etme prosedürlerini belgelemek için kullanılabilir. Kimlik doğrulama bileşenleri durumunda, örneğin, güvenlik birimi testleri, bir hesap kilitlemesi belirlemenin işlevselliğini ve kullanıcı girdi parametrelerinin hesap kilitlemesini atlamak için kötüye kullanılamayacağını (örneğin, hesap kilitlemesini negatif bir sayıya göre ayarlayarak) ileri sürebilir.

Bileşen düzeyinde, güvenlik birimi testleri, hata ve istisna işleme gibi olumsuz iddiaların yanı sıra pozitif iddiaları doğrulayabilir. İstisnalar, sistemi güvensiz bir durumda bırakmadan, tahsis edilmemesinden kaynaklanan potansiyel hizmet reddi (örneğin, nihai bir ifade bloğu içinde kapatılmayan bağlantılar) ve ayrıcalıkların

potansiyel yükseltilmesi (örneğin, istisna atılmadan önce edinilen daha yüksek ayrıcalıklar ve işlevden çıkmadan önce önceki seviyeye ayarlanmamak gibi) gibi engelsiz bir durumda yakalanmamalıdır. Güvenli hata işleme, bilgilendirici hata mesajları ve istif izleri aracılığıyla potansiyel bilgi açıklamasını doğrulayabilir.

Ünite seviyesi güvenlik test durumları, yazılım güvenliği konusunda konu uzmanı olan ve kaynak kodundaki güvenlik sorunlarının giderildiğini ve entegre sistem yapısına kontrol edilebileceğini doğrulamaktan da sorumlu olan bir güvenlik mühendisi tarafından geliştirilmiştir. Tipik olarak, uygulama inşaatlarının yöneticisi, üçüncü taraf kütüphanelerin ve yürütülebilir dosyaların, uygulama yapısına entegre edilmeden önce potansiyel güvenlik açıkları için güvenlik açısından değerlendirilmesini sağlar.

Güvensiz kodlamada kök nedenleri olan ortak güvenlik açıkları için tehdit senaryoları da geliştiricinin güvenlik testi kılavuzunda belgelenebilir. Kaynak kod analizi ile tanımlanan bir kodlama kusuru için bir düzeltme uygulandığında, örneğin, güvenlik testi durumları, kod değişikliğinin uygulanmasının, güvenli kodlama standartlarında belgelenen güvenli kodlama gereksinimlerini takip ettiğini doğrulayabilir.

Kaynak kodu analizi ve birim testler, kod değişikliğinin daha önce tespit edilen kodlama kusurunun maruz kaldığı güvenlik açığını hafiflettiğini doğrulayabilir. Otomatik güvenli kod analizinin sonuçları, sürüm kontrolü için otomatik check-in kapıları olarak da kullanılabilir, örneğin, yazılım eserleri yüksek veya orta şiddet kodlama sorunları ile yapıya kontrol edilemez.

Functional Testers' Security Tests (Fonksiyonel Testlerin Güvenlik Testleri)

Security Testing During the Integration and Validation Phase: Integrated System Tests and Operation Tests (Entegrasyon ve Doğrulama Aşamasında Güvenlik Testi: Entegre Sistem Testleri ve Operasyon Testleri)

Entegre sistem testlerinin temel amacı, "derinlik savunma" kavramını doğrulamak, yani güvenlik kontrollerinin uygulanmasının farklı katmanlarda güvenlik sağlamasıdır. Örneğin, uygulama ile entegre bir bileşen ararken giriş doğrulaması olmaması genellikle entegrasyon testi ile test edilebilecek bir faktördür.

Entegrasyon sistemi test ortamı aynı zamanda testçilerin, uygulamanın kötü niyetli bir dış veya iç kullanıcısı tarafından potansiyel olarak gerçekleştirilebileceği gibi gerçek saldırı senaryolarını simüle edebilecekleri ilk ortamdır. Bu seviyedeki

güvenlik testleri, güvenlik açıklarının gerçek olup olmadığını ve saldırganlar tarafından istismar edilip edilemeyeceğini doğrulayabilir. Örneğin, kaynak kodunda bulunan potansiyel bir güvenlik açığı, potansiyel kötü niyetli kullanıcılara maruz kalmanın yanı sıra potansiyel etki nedeniyle (örneğin, gizli bilgilere erişim) nedeniyle yüksek risk olarak değerlendirilebilir.

Gerçek saldırı senaryoları hem manuel test teknikleri hem de penetrasyon test araçları ile test edilebilir. Bu tip güvenlik testleri de etik bilgisayar korsanlığı testleri olarak adlandırılır. Güvenlik testi perspektifinden bakıldığında, bunlar risk odaklı testlerdir ve uygulamayı operasyonel ortamda test etme amacına sahiptir. Hedef, uygulamaya dönüştürülen uygulamanın üretimine devredilen versiyonunu temsil eden uygulama yapısıdır.

Entegrasyon ve doğrulama aşamasında güvenlik testi de dahil olmak üzere, bileşenlerin entegrasyonu nedeniyle güvenlik açıklarını belirlemek ve bu tür güvenlik açıklarının maruziyetini doğrulamak için kritik öneme sahiptir. Uygulama güvenliği testi, güvenlik mühendislerinin tipik olmadığı hem yazılım hem de güvenlik bilgisi de dahil olmak üzere özel bir beceri seti gerektirir. Sonuç olarak, kuruluşların genellikle yazılım geliştiricilerini etik bilgisayar korsanlığı teknikleri ve güvenlik değerlendirme prosedürleri ve araçları konusunda güvenlikle eğitmeleri gerekir. Gerçekçi bir senaryo, bu tür kaynakları şirket içinde geliştirmek ve bunları geliştiricinin güvenlik testi bilgisini dikkate alan güvenlik test kılavuzlarında ve prosedürlerinde belgelemektir. Örneğin, “güvenlik testi kılıfları hile sayfası veya kontrol listesi” olarak adlandırılan bir, örneğin, sahtecilik, bilgi açıklamaları, tampon taşmaları, format taşmaları, format dizeleri, SQL enjeksiyonu ve XSS enjeksiyonu, XML, SOAP, kanoikleşme sorunları, hizmet reddi ve yönetilen kod ve ActiveX (örneğin, vb. NET) Bu testlerin ilk pili, yazılım güvenliği konusunda çok temel bir bilgi ile manuel olarak gerçekleştirilebilir.

Güvenlik testlerinin ilk amacı, bir dizi asgari güvenlik gereksiniminin doğrulanması olabilir. Bu güvenlik testi durum kılıfları, uygulamayı manuel olarak hataya ve istisnai durumlara zorlamak ve uygulama davranışından bilgi toplamaktan oluşabilir. Örneğin, SQL enjeksiyon güvenlik açıkları, kullanıcı girişi yoluyla saldırı vektörlerini enjekte ederek ve SQL istisnalarının kullanıcıya geri atılıp atılmadığını kontrol ederek manuel olarak test edilebilir. Bir SQL istisna hatasının kanıtı, istismar edilebilecek bir güvenlik açığının bir tezahürü olabilir.

Daha derinlemesine bir güvenlik testi, testçinin özel test teknikleri ve araçları hakkında bilgisini gerektirebilir. Kaynak kodu analizi ve penetrasyon testinin yanı sıra, bu teknikler örneğin: kaynak kodu ve ikili arıza enjeksiyonu, arıza yayılımı analizi ve kod kapsamı, fuzz testi ve ters mühendislik. Güvenlik test kılavuzu, bu tür derinlemesine güvenlik değerlendirmelerini gerçekleştirmek için güvenlik testçileri tarafından kullanılabilecek prosedürleri sağlamalı ve gerekli araçlar önermelidir.

Entegrasyon sistemi testlerinden sonra bir sonraki güvenlik testi seviyesi, kullanıcı kabul ortamında güvenlik testleri yapmaktır. Operasyonel ortamda güvenlik testleri yapmanın benzersiz avantajları vardır. Kullanıcı kabul testi (UAT) ortamı, veriler hariç (örneğin, test verileri gerçek verilerin yerine kullanılır) sürüm yapılandırmasının en çok temsilcisidir. UAT'de güvenlik testinin bir özelliği, güvenlik yapılandırma sorunları için test etmektir. Bazı durumlarda bu güvenlik açıkları yüksek riskleri temsil edebilir. Örneğin, web uygulamasını barındıran sunucu minimum ayrıcalıklar, geçerli SSL sertifikası ve güvenli yapılandırma, devre dışı bırakılan temel hizmetler ve web köknarı dizini test ve yönetim web sayfalarından temizlenmeyebilir.

2.11 Security Test Data Analysis and Reporting (Güvenlik Test Verileri Analizi ve Raporlama)

Goals for Security Test Metrics and Measurements (Güvenlik Test Metrikleri ve Ölçümleri için hedefler)

Güvenlik test metrikleri ve ölçümleri için hedeflerin tanımlanması, risk analizi ve yönetim süreçleri için güvenlik testi verilerinin kullanılması için bir ön koşuldur. Örneğin, güvenlik testlerinde bulunan toplam güvenlik açığı sayısı gibi bir ölçüm, uygulamanın güvenlik duruşunu ölçebilir. Bu ölçümler aynı zamanda yazılım güvenliği testi için güvenlik hedeflerini belirlemeye yardımcı olur, örneğin, uygulama üretime alınmadan önce güvenlik açığı sayısını kabul edilebilir bir minimum sayıya indirir.

Bir başka yönetilebilir hedef, uygulama güvenliği duruşunu, uygulama güvenliği süreçlerindeki iyileştirmeleri değerlendirmek için bir temel çizgiyle karşılaştırmak olabilir. Örneğin, güvenlik metrikleri taban çizgisi, yalnızca penetrasyon testleriyle test edilen bir uygulamadan oluşabilir. Kodlama sırasında da test edilen bir

uygulamadan elde edilen güvenlik verileri, temel çizgiye kıyasla bir iyileşme (örneğin, daha az güvenlik açığı) göstermelidir.

Geleneksel yazılım testlerinde, bir uygulamada bulunan hatalar gibi yazılım kusurlarının sayısı yazılım kalitesinin bir ölçüsünü sağlayabilir. Benzer şekilde, güvenlik testi bir yazılım güvenliği ölçüsü sağlayabilir. Kusur yönetimi ve raporlama perspektifinden, yazılım kalitesi ve güvenlik testi, kök nedenleri ve kusur iyileştirme çabaları için benzer kategorizasyonlar kullanabilir. Kök neden perspektifinden bakıldığında, bir güvenlik kusuru tasarımdaki bir hatadan (örneğin, güvenlik kusurları) veya kodlamadaki bir hatadan (örneğin, güvenlik hatası) nedeniyle olabilir. Bir kusuru düzeltmek için gereken çabanın perspektifinden bakıldığında, düzeltmeyi, gerekli araçları ve kaynakları ve düzeltmeyi uygulama maliyetini uygulamak için geliştirici saatleri açısından hem güvenlik hem de kalite kusurları ölçilebilir.

Güvenlik testi verilerinin bir özelliği, kalite verilere kıyasla, tehdit, kırılganlığın maruz kalması ve riski belirlemeye karşı güvenlik açığının ortaya çıkması açısından kategorizasyondur. Güvenlik için test başvuruları, uygulama karşı önlemlerinin kabul edilebilir seviyeleri karşıladığından emin olmak için teknik riskleri yönetmekten oluşur. Bu nedenle, güvenlik testi verilerinin SDLC sırasında kritik kontrol noktalarındaki güvenlik riski stratejisini desteklemesi gerekir. Örneğin, kaynak kodu analizi ile kaynak kodlarında bulunan güvenlik açıkları, riskin ilk ölçüsünü temsil eder. Güvenlik açığı için bir risk ölçüsü (örneğin, yüksek, orta, düşük) maruz kalma ve olasılık faktörlerini belirleyerek ve penetrasyon testleriyle güvenlik açığını doğrulayarak hesaplanabilir. Güvenlik testlerinde bulunan güvenlik açıklarıyla ilişkili risk metrikleri, risklerin kuruluşun (örneğin, iş ve teknik riskler) farklı seviyelerde kabul edilip edilemeyeceğine, hafifletilemeyeceğine veya aktarılıp aktarılamayacağına karar vermek gibi risk yönetimi kararları almak için işletme yönetimini güvence altına alır.

Bir uygulamanın güvenlik duruşunu değerlendirirken, geliştirilmekte olan uygulamanın boyutu gibi belirli faktörleri göz önünde bulundurmak önemlidir. Uygulama büyüklüğünün, test sırasında uygulamada bulunan sayı sayısıyla ilgili olduğu istatistiksel olarak kanıtlanmıştır. Test sorunları azalttığından, daha büyük boyut uygulamalarının daha küçük boyut uygulamalarından daha sık test edilmesi mantıklıdır.

SDLC'nin çeşitli aşamalarında güvenlik testi yapıldığında, test verileri güvenlik testlerinin tanıtılır getirilmez tespit etme kabiliyetini kanıtlayabilir. Test verileri, SDLC'nin farklı kontrol noktalarında karşı önlemler uygulayarak güvenlik açıklarının giderilmesinin etkinliğini de kanıtlayabilir. Bu tipte bir ölçüm aynı zamanda “kontrol metrikleri” olarak tanımlanır ve geliştirme sürecinin her aşamasında gerçekleştirilen bir güvenlik değerlendirmesinin her aşamada güvenliği koruma yeteneğinin bir ölçüsünü sağlar. Bu koruma metrikleri de güvenlik açıklarını sabitleme maliyetini düşürmede kritik bir faktördür. SDLC'nin aynı aşamasında bulunan güvenlik açıklarıyla başa çıkmak daha ucuzdur, daha sonra başka bir aşamada onları daha sonra düzeltir.

Güvenlik testi metrikleri, aşağıdaki gibi somut ve zamanlanmış hedeflerle ilişkilendirildiklerinde güvenlik riskini, maliyetini ve kusur yönetimi analizini destekleyebilir:

- Güvenlik açıklarının toplam sayısını %30 oranında azaltmak.
- Güvenlik sorunlarını belirli bir son tarihe göre sabitlemek (örneğin, beta sürümünden önce).

Güvenlik testi verileri, manuel kod incelemesi sırasında tespit edilen güvenlik açıklarının sayısı ve penetrasyon testlerine kıyasla kod incelemelerinde tespit edilen güvenlik açıkları sayısı gibi karşılaştırmalı olarak mutlak olabilir. Güvenlik sürecinin kalitesiyle ilgili soruları yanıtlamak için, kabul edilebilir ve iyi olarak kabul edilebilecek şeyler için bir temel belirlemek önemlidir.

Güvenlik testi verileri, güvenlik analizinin belirli hedeflerini de destekleyebilir. Bu hedefler, güvenlik yönetmeliklerine ve bilgi güvenliği standartlarına uygun olabilir, güvenlik süreçlerinin yönetimi, güvenlik kökü nedenlerinin tanımlanması ve süreç iyileştirmeleri ve güvenlik maliyeti fayda analizi olabilir.

Güvenlik test verileri bildirildiğinde, analizi desteklemek için metrikler sağlamak zorundadır. Analizin kapsamı, üretilen yazılımın güvenliği ve sürecin etkinliği hakkında ipuçları bulmak için test verilerinin yorumlanmasıdır.

Güvenlik test verileri tarafından desteklenen bazı ipuçları örnekleri şunlar olabilir:

- Açıklar serbest bırakılmak için kabul edilebilir bir seviyeye indirgenir mi?
- Bu ürünün güvenlik kalitesi benzer yazılım ürünleriyle nasıl karşılaştırılır?
- Tüm güvenlik testi gereksinimleri karşılanıyor mu?

- Güvenlik sorunlarının başlıca temel nedenleri nelerdir?
- Güvenlik hatalarına kıyasla güvenlik kusurları ne kadar çok?
- Güvenlik açıkları bulmada en etkili güvenlik faaliyeti en etkilidir?
- Güvenlik kusurlarını ve güvenlik açıklarını düzeltmede hangi ekip daha üretken?
- Genel güvenlik açıklarının yüzde kaçı yüksek risk altındadır?
- Güvenlik açıklarını tespit etmede en etkili hangi araçlardır?
- Ne tür güvenlik testleri güvenlik açıkları (örneğin, beyaz kutuya karşı siyah kutuya karşı) testlerde etkilidir?
- Güvenli kod incelemeleri sırasında kaç güvenlik sorunu bulunur?
- Güvenli tasarım incelemeleri sırasında kaç güvenlik sorunu bulunur?

Test verilerini kullanarak sağlam bir karar vermek için, test araçlarının yanı sıra test sürecini de iyi anlamak önemlidir. Hangi güvenlik araçlarının kullanılacağına karar vermek için bir araç taksonomisi benimsenmelidir. Güvenlik araçları, farklı eserleri hedeflerken yaygın, bilinen güvenlik açıklarını bulmada iyi olarak nitelendirilebilir.

Bilinmeyen güvenlik sorunlarının test edilmediğini belirtmek önemlidir. Bir güvenlik testinin sorunlardan uzak olması, yazılımın veya uygulamanın iyi olduğu anlamına gelmez.

En sofistike otomasyon araçları bile deneyimli bir güvenlik test cihazı için bir eşleşme değildir. Sadece otomatik araçlardan elde edilen başarılı test sonuçlarına güvenmek, güvenlik uygulayıcılarına yanlış bir güvenlik hissi verecektir. Tipik olarak, güvenlik test cihazları güvenlik test metodolojisi ve test araçları ile ne kadar deneyimli olursa, güvenlik testi ve analizinin sonuçları o kadar iyi olacaktır. Güvenlik test araçlarına yatırım yapan yöneticilerin, güvenlik testi eğitiminin yanı sıra yetenekli insan kaynaklarının işe alınmasına da yatırım yapmaları önemlidir.

Reporting Requirements (Raporlama Gereksinimleri)

Bir uygulamanın güvenlik duruşu, güvenlik açıklarının sayısı ve güvenlik açıklarının risk derecesi gibi etki perspektifinden ve kodlama hataları, mimari kusurlar ve yapılandırma sorunları gibi neden veya kökene bakış açısından karakterize edilebilir.

Güvenlik açıkları farklı kriterlere göre sınıflandırılabilir. En yaygın kullanılan güvenlik açığı şiddeti metrik, Olay Tepkisi ve Güvenlik Ekipleri Forumu (FIRT) tarafından sürdürülen bir standart olan Ortak Güvenlik Açığı Puanlama Sistemi (CVSS).

Güvenlik testi verilerini bildirirken, en iyi uygulama aşağıdaki bilgileri içermelidir:

- Her bir güvenlik açığının türe göre bir kategorizasyonu;
- Her bir sorunun maruz kaldığı güvenlik tehdidi;
- Böcek veya kusur gibi her güvenlik sorununun temel nedeni;
- Sorunları bulmak için kullanılan her test tekniği;
- Her bir güvenlik açığı için iyileştirme veya karşı önlem; ve
- Her bir güvenlik açığının şiddet derecesi (örneğin, yüksek, orta, düşük veya CVSS puanı).

Güvenlik tehdidinin ne olduğunu anlatarak, azaltma kontrolünün tehdidi azaltmada etkisiz olup olmadığını ve neden etkisiz olduğunu anlamak mümkün olacaktır.

Sorunun temel nedenini bildirmek, neyin düzeltilmesi gerektiğini belirlemeye yardımcı olabilir. Örneğin, beyaz kutu testi söz konusu olduğunda, güvenlik açığının yazılım güvenliği kökü nedeni rahatsız edici kaynak kodu olacaktır.

Sorunlar bildirildikten sonra, yazılım geliştiricisine nasıl yeniden test edileceği ve güvenlik açığının nasıl bulunacağı konusunda rehberlik etmek de önemlidir. Bu, kodun savunmasız olup olmadığını bulmak için bir beyaz kutu test tekniği (örneğin, statik kod analizörü ile güvenlik kodu incelemesi) kullanmayı içerebilir. Bir kara kutu penetrasyon testi ile bir güvenlik açığı bulunabilirse, test raporunun ön uçtaki (örneğin, müşteri) güvenlik açığının nasıl doğrulanacağı konusunda bilgi vermesi gerekir.

Güvenlik açığının nasıl düzeltileceği hakkındaki bilgiler, bir geliştiricinin bir düzeltme uygulaması için yeterince ayrıntılı olmalıdır. Güvenli kodlama örnekleri, yapılandırma değişiklikleri sağlamalı ve yeterli referanslar sağlamalıdır.

Son olarak, şiddet derecesi risk derecesinin hesaplanmasına katkıda bulunur ve iyileştirme çabasına öncelik vermeye yardımcı olur. Tipik olarak, güvenlik açığına bir risk derecesi atamak, etki ve maruz kalma gibi faktörlere dayanarak dış risk analizini içerir.

Business Cases (İş Davaları)

Güvenlik testi metriklerinin yararlı olması için, kuruluşun güvenlik testi veri paydaşlarına değer sağlamaları gerekir. Paydaşlar arasında proje yöneticileri, geliştiriciler, bilgi güvenliği ofisleri, denetçiler ve baş bilgi görevlileri yer alabilir. Değer, her bir proje paydaşının rol ve sorumluluk açısından sahip olduğu iş durumu açısından olabilir.

Yazılım geliştiricileri, yazılımın güvenli ve verimli bir şekilde kodlandığını göstermek için güvenlik testi verilerine bakar. Bu, kaynak kodu analiz araçlarını kullanmak, güvenli kodlama standartlarını takip etmek ve yazılım güvenliği eğitimine katılmalarını sağlar.

Proje yöneticileri, proje planına göre güvenlik test faaliyetlerini ve kaynaklarını başarılı bir şekilde yönetmelerini ve kullanmalarını sağlayan veriler ararlar. Proje yöneticileri için, güvenlik testi verileri, projelerin programda olduğunu ve teslimat tarihleri için hedefe geçtiğini ve testlerde daha iyi hale geldiğini gösterebilir.

Güvenlik testi verileri, girişimin bilgi güvenliği görevlilerinden (ISO'lar) gelmesi durumunda güvenlik testi için iş davasına da yardımcı olur. Örneğin, SDLC sırasındaki güvenlik testlerinin proje dağıtımını etkilemediğine dair kanıtlar sağlayabilir, bunun yerine üretimde daha sonra güvenlik açıklarını ele almak için gereken genel iş yükünü azaltır.

Uyum denetçilerine göre, güvenlik testi metrikleri, güvenlik standardı uyumluluğunun kuruluş içindeki güvenlik inceleme süreçleri aracılığıyla ele alındığına dair bir dizi yazılım güvenliği güvencesi ve güven sağlar.

Son olarak, güvenlik kaynaklarında tahsis edilmesi gereken bütçeden sorumlu olan Baş Bilgi Görevlileri (CIO'lar) ve Baş Bilgi Güvenliği Görevlileri (CISO'lar), güvenlik test verilerinden maliyet-fayda analizinin türetilmesini arar. Bu, hangi güvenlik faaliyetlerinin ve araçlarının yatırım yapacağı konusunda bilinçli kararlar vermelerini sağlar. Bu tür analizleri destekleyen metriklerden biri de güvenlikte Yatırım Getirisi (ROI). Bu tür metrikleri güvenlik test verilerinden elde etmek için, güvenlik açıklarının maruz kalması ve güvenlik testlerinin güvenlik riskini azaltmadaki etkinliği nedeniyle risk arasındaki farkı ölçmek, daha sonra bu boşluğu güvenlik test faaliyetinin maliyeti veya benimsenen test araçlarıyla faktörize etmek önemlidir.

References (Referanslar)

- ABD Ulusal Standartlar Enstitüsü (NIST) 2002, yetersiz yazılım testi nedeniyle güvensiz yazılımların ABD ekonomisine maliyetine ilişkin anketi