Güvenli Yazılım Geliştirme

Yazan: Yiğit Turak

06 Haziran 2014

İçindekiler

Kısaltmalar ve Tanımlar

Kısaltma / Tanım	Açıklama
Hacker	Bilgisayar ve haberleşme teknolojileri konusundaki bilgisini gizli verilere ulaşmak, ağlar üzerinde yasal olmayan zarar verici işler yapmak için kullanan kişi
Batch	Belirli aralıklarla belirli işlemleri otomatik yapmak için programlanmış işlem
Bottlenack	İşlemlerin ilerlemesini engelleme veya yavaşlatma
Proxy	Bir web tarayıcısı (Internet Explorer gibi) ve Internet arasında aracı işlevi gören sunucu
FTP	Dosya transfer protokolü (File Transfer Protocol)
DNS	Alan adı sunucusu (DomainName Server)

Özet
Yazılımlara olan ihtiyaçların artması ile birlikte, belirli bir sistematik oluşturabilmek için yazılım geliştirme süreçleri oluşturulmuştur. Ancak günümüzde bilişim teknolojilerinin hızla gelişmesi, teknolojik cihazların kullanımının giderek artması, bilginin anında işlenebilmesi ihtiyacı, geliştirilen yazılım sayısının ciddi derecede artmasına sebep olmaktadır. Ayrıca internetin sağladığı imkânlar, yazılım sayısının çok fazla olması, insanların merak duygusunu artırmış ve bu paralelde güvenlik tehditlerinin de artmasını tetiklemiştir. Son 5 yıl içerisinde yaşanan siber olaylar, bireylerde güvenlik algısı oluşturmuştur. Yazılımlarda tespit edilen güvenlik zafiyetlerinin yazılım geliştirme süreci içerisinde kapatılması noktasında ortaya çıkan maliyetleri güvenlik temelli yazılım geliştirme süreçlerinin oluşmasını sağlamıştır. Bununla beraber, bilişim teknolojilerine olan bağımlılığın giderek artması, bu alanda güvenlik algısı oluşmuş çalışan oranının gün geçtikçe azalması, ortaya çıkarılan yazılım ürünlerindeki zafiyetler sebebiyle büyük kurumların ciddi para ve itibar kayıplarına, yazılım geliştirmelerde gerek yazılım süreçlerine gerekse de güvenli yazılım geliştirme süreçlerine gereken önemin verilmemesinden kaynaklandığını düşünmekteyim. Bu alandaki bilgi birikimine katkıda bulunmak amacıyla bu çalışmanın ilk bölümünde yazılım geliştirme sürecini, güvenli yazılım geliştirme sürecini ve maliyete etkisi ele alınacak, ikinci bölümünde güvenlik süreçleri ve yazılım geliştirme sırasında ele alınması gereken temel kontrol noktaları incelenecektir.

Anahtar Kelimeler: Yazılım Geliştirme, Güvenli Yazılım Geliştirme, SDLC, Güvenlik Süreçleri

1 Giriş

Gün geçtikçe sık duymaya başladığımız kelimeler arasında yerini alan **yazılım** sözcüğü Türk Dil Kurumu sözlüğünde "bir problemi çözmek amacıyla farklı cihazların birbirleriyle iletişim kurabilmesini sağlayan ve görevlerini ya da kullanılabilirliklerini geliştirmeye yarayan, bilgisayar dili kullanılarak oluşturulmuş bir ifadedir" şeklinde tanımlanmaktadır.

Günümüzde bilişim teknolojilerinin hızla gelişmesi, teknolojik cihazların kullanımının giderek artması, bilginin anında işlenebilmesi ihtiyacı, geliştirilen yazılım ve yazılım geliştiricilerin sayısının ciddi derecede artmasına sebep olmuştur. Uluslararası Veri Birliği'nin (IDC) ön görülerine göre 2014 yılı içerisinde yaklaşık 29 milyon kişinin bilgi ve iletişim teknolojileri alanında yetkin kişi, 18 milyon da yazılım geliştirici olacağı yönündedir. Ayrıca dünya nüfusunun yaklaşık üçte ikisinin internet bağlantısı, %20'sinin sosyal ağlara üyelikleri ve Türkiye nüfusunun %49'unun aktif internet erişimi bulunmaktadır. Yine dünya nüfusunun %85'i cep telefonu kullanmakta ve bunların %15'i cep telefonlarıyla alışveriş yapmaktadırlar. Bu rakamlar bilişim teknolojilerine dolayısıyla yazılımlara olan bağımlılığın ne derece arttığını göstermekte olup, yazılım geliştirme süreçlerinin oluşturulması ve derinlemesine araştırılması yadsınamaz bir gerçek olmuştur.

Bilişim teknolojileri ve yazılımların, hayatı kolaylaştırma adına sağladıkları imkânların yanında, güvenlik boyutunda da yeni kaygıların gelişmesine sebep olmuştur. Birçok uluslararası firmanın geliştirdiği yazılımlar hackerlar karşısında direnememiş ve şirketler bu durumdan çok ciddi zararlar görüşmüştür. Örneğin Sony firmasının PlayStation ürünü kullanıcılarının çevrimiçi oyun oynadıkları yazılımın kırılması sonucu şirketin kaybı yaklaşık 150 milyon £ zarara uğramıştır. Ayrıca yapılan araştırmalar sonucu tespit edilen güvenlik zafiyetlerinin %80 gibi bir oranı yazılım kaynaklıdır. Bununla birlikte gün geçtikçe bilgi teknolojileri alanında çalışan insanlarda artan güvenlik algısı, yazılım geliştirme süreçlerinin içerisine güvenlik süreçlerinin de dâhil edilmesine ve yeni süreçlerin oluşmasına sebep olmuştur.

Bu çalışmadaki ana amaç yazılım geliştirme ve güvenli yazılım geliştirme süreçlerini incelemektir. Çalışmanın ilk bölümünde, yazılım geliştirme süreci, güvenli yazılım geliştirme süreci ve yazılım-güvenlik-maliyet tablosundan bahsedilmiştir. Çalışmanın ikinci bölümünde ise güvenlik süreçlerine ve yazılım geliştirme sırasında dikkat edilmesi gereken temel güvenlik kontrollerine yer verilmiştir.

2 (GÜVENLİ) YAZILIM GELİŞTİRME

Gün geçtikçe daha fazla duymaya başladığımız yazılım kavramı için iş ihtiyaçlarına göre süreçler belirlenmek zorunda kalınmıştır. Bu süreçlerden temel yazılım geliştirme ve güvenli yazılım geliştirme süreçlerini ve özelliklerini bu bölümde inceleyeceğiz.

2.1 YAZILIM GELİŞTİRME SÜRECİ

Geçmişte yazılım geliştirmede başvurulan iş akış şemaları gibi yöntemler günümüzde gereksinimleri karşılayamadıklarından etkinliklerini yitirmişlerdir. Yazılım işlevleri ile ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için, söz konusu aşamalar sürekli bir döngü biçiminde ele alınmaktadır. Böylece döngü içerisinde herhangi bir aşamada geriye dönmek ve tekrar ilerlemek söz konusudur. Yazılım geliştirmede çok sayıda farklı model ve süreç değerlendirmelerinden söz etmek mümkündür. Bununla birlikte; yazılım mühendisliğindeki diğer modellere temel teşkil eden "Çağlayan Modeli (Waterfall Model)" yazılım yaşam döngüsünü analiz, tasarım, geliştirme, test ve dağıtım olmak üzere beş aşamada ele almaktadır.

2.1.1 Analiz

Yazılımların ne yapacağını ve nasıl yapacağını belirlediğimiz, problemi tanımladığımız aşamadır. Yazılım ancak isteneni doğru bir biçimde yerine getiriyorsa başarılı bir yazılımdır. Bu sebeple öncelikle yazılımdan ne istendiğinin doğru bir biçimde tanımlanması gerekir. Analiz aşaması personel, donanım ve sistem gereksinimlerinin belirlenmesi, sistemin fizibilite çalışmasının yapılması, kullanıcıların gereksinimlerinin analizi, sistemin ne yapıp ne yapmayacağının kısıtlamalar göz önüne alınarak belirlenmesi, bu bilginin kullanıcılar tarafından doğrulanması ve proje planı oluşturulması adımlarından oluşur.

2.1.2 Tasarım

Yazılım tasarımı, bir bileşen veya sistemin nasıl gerçekleştirileceğini belirlemek için kullanılan teknikler, stratejiler, gösterimler ve desenlerle ilgilidir. Bu aşama mimari tasarım, veri tasarımı, kullanıcı ara yüzü tasarımı, tasarım araçları ve tasarımın değerlendirilmesi alt süreçlerini kapsamaktadır. Tasarım aşaması, yazılımın hem kullanıcı ara yüzünü hem de programın omurgasını ortaya koymaktadır. Yapılacak tasarım, yazılımın işlevsel gereksinimlere uygun olmasının yanı sıra kaynaklar, performans ve güvenlik gibi kavramları da göz önüne alınarak gerçekleştirilmelidir.

2.1.3 Geliştirme

Geliştirme aşaması, tasarım sürecinde ortaya konan veriler doğrultusunda yazılımın gerçekleştirilmesi aşamasıdır. Tasarım sonucu üretilen sürecin, bilgisayar ortamında çalışan yazılım biçimine dönüştürülmesi çalışması olarak da nitelendirilebilir. Yazılım geliştirme ortamı, programlama dili, veri tabanı yönetim sistemi, yazılım geliştirme araçları seçimi kodlama aşamasında gerçekleştirilir.

2.1.4 Test

Test aşamaları ise Alfa testi, Beta testi ve Kullanıcı Kabul testidir. Alfa testi yazılımcı tarafından geliştirme yaptığı uygulamanın çalışıp çalışmadığı kontrolünün yapıldığı süreçtir. Beta testi, iş analistleri ve proje yönetimindeki kişiler tarafından yapılan yüzeysel fonksiyonellik kontrolü yapılan testlerdir. Kullanıcı Kabul testi ise müşteriler tarafından yazılımın tüm fonksiyonelliklerini belirli senaryolar ışığında yapılmasıdır.

2.1.5 Dağıtım

Test aşaması sonrası ilgili ekiplerden onay alan yazılımın üretim ortamına aktarıldığı aşamadır. Ayrıca yazılımın kullanıma başlanmasından sonra yazılımın desteklenmesi sürecini de kapsar. Yazılımın eksiklerinin giderilmesi, iyileştirilmesi gibi alt aşamaları içeren aşamadır.

2.2 GÜVENLİ YAZILIM GELİŞTİRME

Gün geçtikçe yazılım/programlama gibi kavramların daha çok hayatımızın içerisine girmesi, yeni saldırı tiplerini doğurmakta olup, yeni yazılım geliştirme yöntemlerini ve yeni programlama kontrol noktalarını ortaya çıkarmaktadır. İş sahiplerinin ve yazılım geliştirme ekiplerinin çok hızlı şekilde yeni ürünü yayınlamak istemesi, kaliteyi düşürdüğü gibi birçok güvenlik noktasının da ihlal edilmesine sebep olmaktadır. Son günlerde artan ve artmaya da devam eden güvenlik algısı sebebiyle yazılım geliştirme süreçleri de yeniden tasarlanmış ve güvenlik temel nokta alınmıştır. Bu bölümde güvenli yazılım geliştirme sürecinin temel etaplarını inceleyeceğiz.

2.2.1 Gereksinim & Risk Analizi

Güvenli yazılım geliştirme sürecinin en temel aşaması olup yaşam döngüsü gereksinim aşaması ile başlar. Gereksinim aşamasının amacı; müşterinin taleplerini yerine getiren yazılım gereksinimlerini belirlemek ve aynı zamanda açık ve anlaşılır bir şekilde temsil etmektir. Yazılım ihtiyaçlarının belirlenmesi ile birlikte, çalışmalar güvenlik gereksinimlerini belirlemek için yapılır. Yazılıma ilgili risk birimleriyle birlikte risk analizi yapılarak güvenlik için gereken minimum gereksinimler belirlenir. Güvenlik gereksinimleri farklı kaynaklardan farklı zamanlarda ortaya çıkabilmektedir. Örneğin; bazı güvenlik gereksinimleri, tasarım aşamasında tehdit modellemeden sonra tanımlanır. Tanımlama yapıldıktan sonra, güvenlik gereksinimleri diğer yazılım gereksinimleriyle birlikte analize tabi tutulur. Analizden sonra, gereksinimlerin dokümantasyonu yapılır.

2.2.2 Tasarım & Tehdit Modelleme

Tasarım aşaması yazılım mimarisinin oluşturulması ile başlayıp; kullanıcının doğrudan maruz kalacağı güvenlik esaslarını, fonksiyonların güvenli bir şekilde nasıl

uygulanacağının belirlendiği tehdit modelleme ile tamamlanır. Tasarım aşaması boyunca güvenlik ve gizlilik kaygılarını dikkatle almak çok önemlidir. Güvenlik mimarisi de yazılım mimarisinden geliştirilmiştir. Tasarım aşamasında ilk olarak etkileşim noktaları tanımlanır. Bu noktalar yazılımın iş akışını anlamada yardımcı olur ve ortaya çıkabilecek güvenlik sorunları hakkında derinlemesine bilgi verir. Güvenlik tasarımındaki temel amaç saldırı yüzeyini en aza indirgemektir. Bu aşama saldırganın çalışma alanını azaltmanın en etkili yoludur. Yazılım tasarımı ayrıntılı olarak analiz edilir ve maksimum kullanıcı sayısının ihtiyaçlarını etkilemeden bir hacker gibi düşünerek etki altında kalan yüzey alanının azaltılması konusunda çalışma yapılır. Tehdit modelleme güvenli yazılım aşamasında önemli bir kilometre taşı olarak kabul edilebilir. Tehdit modelleme; uygulamaları bir yazılımın nasıl saldırıya maruz kalabileceğine, neyin saldırabileceğine, hangi alanlarda saldırı eğilimi olabileceğine, ne tür atakların uygulanabileceğine dair tam bir bilgi sağlar. Bu bilgiler ışığında, tehdit modelleme sürekli olarak güncellenir.

2.2.3 Geliştirme & Kaynak Kod Analizi

Güvenlik perspektifinden bakıldığında bu aşama 2 ana bölümden oluşmaktadır. İlk aşama güvenli kod yazarak gerçekleştirilir. Güvenli kod yazmak oldukça önemlidir, çünkü yapılmadığı takdirde bütün çabalar boşa gidecektir. Güvenli kod, tasarım aşamasında tanımlanan tehdit modelindeki tehditler ve genel güvenlik kuralları dikkate alınarak yazılır. Tüm bu faktörler, güvenli kod yazmak için çok önemlidir ve herhangi birinin eksikliği güvenlik açıklarına sebep olabilir. İkinci aşama ise güvenlik açıklarını tespit etmek için statik analizin kullanılması ve çıkan sonuca göre yazılım üzerinde gerekli güncellemeleri yapmaktır. Geliştirme ekibi tarafından yazılmış kod, geliştirme ekipleri tarafından otomatik araçlar yardımı ile analiz edilir. Tarama sırasında tespit edilen güvenlik açıklarına göre yazılım ekipleri gerekli güncellemeleri yeniden yapar ve tekrar taramadan geçirirler. Bu döngü güvenlik açığı çıkmayana kadar devam etmelidir. Bu araçların seçerken ve kullanırken temel noktalar, güvenlik ekibi tarafından kullanılması onaylanmış olması ve en güncel sürümün kullanılmasıdır.

2.2.4 Test

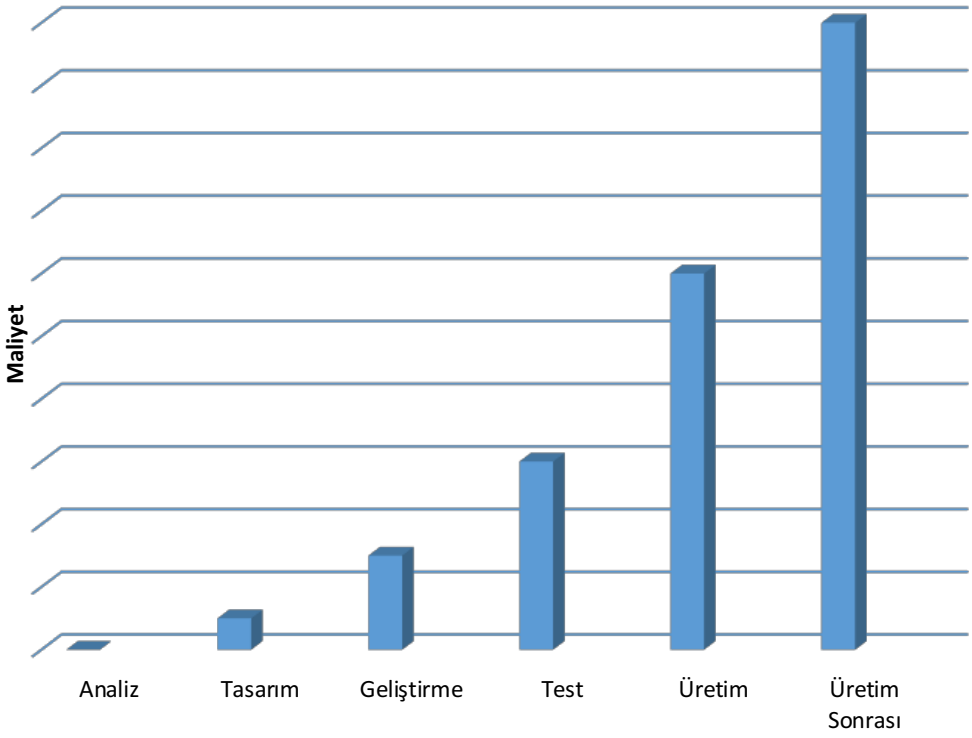
Geliştirilen uygulama yayınlanmadan önce doğrulama ve test aşamalarından geçirilmelidir. Bu aşamalar temelde güvenlik ve test olarak ikiye ayrılır. Güvenlik aşamaları ise statik kod analizi, sızma testi ve stres testidir. Yazılım ekipleri tarafından güvenlik açığı kalmamış yazılım son bir kez de güvenlik uzmanları tarafından statik kod analizinden geçirilir. Eğer bulgu tespit edilirse, bulgunun kapatılması için tekrar geliştirme aşamasına dönülür. Sızma testi ile uygulama üzerinde kod analizinde tespit edilememiş zafiyetler taranır. Stres testi ile yazılımların dayanıklılık ve sınırları tespit edilir.
Test aşamaları ise Alfa testi, Beta testi ve Kullanıcı Kabul testidir. Alfa testi yazılımcı tarafından geliştirme yaptığı uygulamanın çalışıp çalışmadığı kontrolünün yapıldığı süreçtir. Beta testi, iş analistleri ve proje yönetimindeki kişiler tarafından yapılan yüzeysel fonksiyonellik kontrolü yapılan testlerdir. Kullanıcı Kabul testi ise müşteriler tarafından yazılımın tüm fonksiyonelliklerini belirli senaryolar ışığında yapılmasıdır.

2.2.5 Dağıtım & Monitör

Yazılımın geliştirme test aşaması bittiğinde, son kullanıcıya dağıtma başlanmadan hemen önce geliştirilen yazılımın dağıtımı yapılacak hali ile gerekli testlerden başarıyla geçmiş halinin aynı olduğundan emin olunması gerekmektedir. İlgili uzmanların hepsinin uygunluğu alındıktan sonra yazılım artık üretim ortamına aktarılır. Daha sonrasında, iş sahibi, yazılım geliştirme ekibi ve güvenlik uzmanları yazılımı yeni çıkan güvenlik tehditlerine karşı izlemeye başlar. Tespit edilen açıkları yazılım geliştirme ve güvenlik ekipleri değerlendirir. Değerlendirme sonucunda güvenli yazılım geliştirme sürecine uygun olarak yeni sürümler hazırlanır ve dağıtımı yapılır.

2.3 YAZILIM-GÜVENLİK-MALİYET ÜÇGENİ

Geliştirilen yazılımların gün geçtikçe artması yazılımlara ilişkin çalışmaları hızlandırarak yeni yazılım geliştirme yöntemlerini, yeni programlama kurallarını ve derleyici çeşitlerini ortaya çıkarmaktadır. Geliştiricilerin piyasaya hızlı ürün sürme isteği ve hatalı kod geliştirilmesi güvenilirliği sağlayamama ve düşük kalite gibi sorunlara yol açmaktadır. Araştırmalara göre bilişim güvenliği ihlallerinin %80 gibi büyük bir bölümü yazılım güvenliğinin eksikliğinden oluşmaktadır. Genel olarak problemlerin çoğu, yazılım geliştirme sürecinde gereksinim ve sistem analizlerinin doğru ve yeterli yapılmamasından kaynaklanmaktadır. Yazılımlardaki analiz eksikliği güvenlik riski oluşturmakta, bu durum bilgiye yönelik tehditlerin ortaya çıkmasında önemli bir açıklık oluşturmaktadır. Yazılım geliştirmenin başlarında farkına varılan yazılım açıklıklarının düzeltilmesinin daha ileri süreçlerde farkına varılan açıklıklara göre daha az maliyetli olacağı kabul edilmiştir. Aşağıdaki Şekil 1 de yazılım geliştirme sürecinde, yazılım açıklarının maliyet analizi gösterilmektedir.



Şekil 1 Yazılım-Güvenlik-Maliyet Üçgeni

3 GÜVENLİK SÜREÇLERİ

Yazılım geliştirilirken her zaman en önemli unsur yazılımın yapması gereken işi bir an önce yapmasını sağlamak olmuştur. Sağlam, dirençli, dayanıklı, güvenilir ve güvenli yazılımlar geliştirmek ikinci plana atılmaktadır.
Bir yazılımın güvenli olup olmadığı için belirli güvenlik süreçlerinden geçirilmesi gerekmektedir. Ayrıca yazılım geliştirme sırasında bazı kontrol noktaları dikkate alınmalıdır.

3.1 GÜVENLİK TARAMALARI

3.1.1 Statik Kod Analizi

Bir yazılımı çalıştırmadan yazılımın kaynak kodunda yer alan olası hataları bulmak ve yazılımın kodlama kurallarına uygunluğunu test etmek için kullanılan yöntemler **statik kod analizi** olarak adlandırılır. Bir uygulamanın olası her türlü senaryoyu kapsayacak şekilde test edilmesi mümkün olmayabilir. Ancak uygulama kaynak kodu incelendiğinde uygulamanın tamamının herhangi bir bölüm, senaryo atlanmadan incelenmesi ve açıkların tespit edilmesi mümkün olabilir. Örneğin bir web uygulamasında sızma testi ile sadece belirli aralıklarda çalışan (batch) bir işlevdeki açıkların tespit edilmesi mümkün olmayacağı halde kaynak kod analizi ile bu açıkların da tespit edilmesi mümkün olacaktır. Kaynak kod incelemesi ile uygulamalarda başka bir şekilde tespit edilmesi mümkün olmayabilecek mantık hataları, art niyetli kod bölümleri gibi sorunlar da tespit edilebileceklerdir.
Statik kod analizi 2 farklı şekilde yapılmaktadır:

1. **Temel kod inceleme:** Otomatik araçlarla gerçekleştirilen kod analizi çalışmasından elde edilen sonuçlar uzman kişiler tarafından değerlendirilerek gerekli aksiyonlar alınır.

2. **Detaylı kod inceleme:** Otomatik araçların kullanılmasının yanı sıra uygulama geliştirme tecrübesine sahip gerçek uzmanlar tarafından uygulama tasarımı ve kaynak kodu incelenir, tespit edilen açıkların giderilmesi için detaylı çözüm önerileri sunulur ve aksiyonlar alınır.

3.1.2 Sızma Testi

Genel anlamda iç ağ veya internet üzerinden erişilebilir kurum kaynaklarına (dns, ftp, e-posta, web, güvenlik duvarı, sunucular vb.) isteğe bağlı olarak yetkili veya yetkisiz kullanıcı hakları ile değişik araçlar ve yöntemler kullanılarak gerçekleştirilen ve bilinen muhtemel güvenlik açıklarını saldırganlardan önce keşfetmeyi amaçlayan testlerdir.
Yazılım geliştirme sürecinde yapılması gerekli olan sızma testindeki amaçlar ise kaynak kod taraması sırasında tespit edilemeyen uygulama, yetkilendirme veya sunucu bazlı zafiyetleri tespit etmek amaçlanır. Uzman kişiler tarafından yapılan test sonucunda zafiyeti ve çözüm önerileri sununa bir rapor hazırlanır; ilgili aksiyonlar alınır.

3.1.3 Stres Testi

Yazılımların temelde dayanıklılıklarını test etmek, sınırlarını tespit etmek için kullanılan bir metottur. Belirlenen kısıtlar ölçüsünde uygulanan yük altında, sistemde ortaya çıkabilecek darboğazları (bottleneck) ortaya çıkarmak; uygulamaya dönük bir açık, hata gibi normal çalışma şartlarında ortaya çıkmayan, ama ağır yüklenmeler karşısında ortaya çıkabilecek sonuçlar oluşturmak ve çok sayıda veri girişi, büyük nümerik değerler, çok sayıda sorgu kullanarak uygulamanın dayanıklılığını belirlemektir.

3.2 TEMEL KONTROL NOKTALARI

3.2.1 Girdi Kontrolü

Günümüzde bilinen güvenlik zafiyetlerinin çoğu girdi kontrollerinin tam anlamıyla yeterli seviyede yapılmamasından kaynaklanmaktadır. Girdi kontrol yöntemleri "beyaz liste" (whitelist) ve "kara liste" (blacklist) olmak üzere ikiye ayrılmaktadır. Beyaz liste yönteminde bilinen bir şablon girdi olarak kullanılmakta, bu şablonun dışındaki tüm girdiler kötü niyetli olarak kabul edilmektedir. Şablonun kontrolü çok kolay olduğundan bu yöntem oldukça etkili bir yöntemdir. Kara liste yöntemi ise daha az etkili olmasına rağmen daha çok tercih edilen bir yöntemdir. Bu yöntemde kullanılan belirli bir şablon yoktur, sadece bilinen saldırıların bir listesi mevcuttur. Eğer girdi bilinen bir saldırıya benziyor ise o zaman girdi reddedilecek, onun dışındaki tüm girdiler ise kabul edilecektir. Bugün bile tüm atak çeşitlerini belirlemek zor iken gelecekteki atakları bilip filtrelemek daha da zor olacağından bu yöntemin etkinliğinin az olduğu açıktır. Dolayısıyla veri yapıları, mümkün olduğunca belli bir şablona uygun tasarlanarak kontrolü daha güçlü kılınmalıdır.
İstemci-sunucu uygulamalarında girdi kontrolleri hem istemci hem de sunucu tarafında yapılabilmektedir. Bununla birlikte; bir saldırgan istemci tarafındaki kontrolü kolay aşabileceğinden istemci tarafındaki kontrol hiçbir zaman yeterli bir güvenlik önlemi olarak görülmemeli ve girdi kontrolü iki tarafta da kesinlikle yapılmalıdır. Kısaca güvenilir olmayan bir kaynaktan (örneğin kullanıcıdan) gelen veri mutlaka onaylanmalıdır.

3.2.2 Kimlik Doğrulama ve Yetkilendirme

Kimlik doğrulama, varlıkların (kullanıcı, cihaz veya bir uygulama) kimlik kontrolünden geçmesi işlemi olup, bu işlem farklı kimlik doğrulama yöntemleri ile yapılabilmektedir. Geçmişte uygulamalar sadece kullanıcı adı ve şifre kullanması şeklinde zayıf doğrulama yöntemleri kullanılmakta iken, günümüzde bir "domain" yapısı varsa, kullanıcılar "Active Directory" kullanılarak doğrulanmakta, "domain" dışında ise kimlik yönetimine ilişkin veri tabanı veya sertifikalar uygulanmaktadır. Daha güçlü doğrulama yöntemleri olarak da biometrik metotlar veya akıllı kartlar kullanılmaktadır. Bir diğer doğrulama yöntemi ise üçüncü bir tarafın doğrulama işini yapması ve bu üçüncü tarafa güven duyulması şeklindedir.

Kullanıcıların tanımlanması aşaması olan kimlik doğrulamadan sonra kullanıcının kimliği doğrultusunda erişim haklarının belirlendiği ve kontrolünün gerçekleştiği aşama yetkilendirmedir. Hangi yetkilerle işlem yapılacağını belirlemek için birçok yöntem bulunmaktadır. Yetkilendirme konusunda uygulamalar üzerinde çeşitli profiller belirlenmeli, bu profillere sahiplikler atanmalı ve bir kullanıcı bir profil için yetki talep ettiği zaman talep ettiği profilin sahibinden onay alınmalıdır. Ayrıca profillerin sahipleri yıllık olarak yetkilendirme matrislerini gözden geçirmelidir.

3.2.3 Konfigürasyon Yönetimi

Konfigürasyon, uygulama ile ilgili hassas bilgileri içermektedir. Örnek vermek gerekirse veri tabanına erişim için gerekli bağlantı bilgilerini içeren dosyalar bu kapsamdadır. Konfigürasyona müdahale uygulamanın işleyişini değiştirebilir veya çalışmamasına sebep olabilir. Konfigürasyon dosyalarının sunucularda saklanıyor olması yeterli güvenlik önlemlerinin alındığı anlamına gelmemektedir. Konfigürasyon dosyaları hassas bilgi olarak nitelendirilmeli, şifrelenmiş bir şekilde tutulmalı ve bu dosyalara erişim kayıt altında tutulmalıdır.

3.2.4 Hassas Bilgi

Hassas bilginin ne olduğunun belirlenebilmesi için uygulamanın ve işin bir arada ele alınması gerekir. Uygulama geliştirici işin niteliğini tam olarak bilemediğinden, diğer yandan işin sahibi de uygulamanın teknik altyapısı hakkında sınırlı bilgiye sahip olacağından bu iki taraf tek başlarına hassas bilgi için yeterli tanımlama yapamayacaklardır. İki tarafın bir araya gelmesiyle hassas bilgileri içeren bir liste oluşturulmalı ve bu listeyi koruyacak bir politika oluşturulmalıdır. Örneğin anne kızlık soyadı, kredi kartı numarası gibi bilgiler uygulama ekranında da veri tabanında da maskeli olarak gösterilmelidir.

3.2.5 Kriptografi

Veriyi korumanın yollarından biri de şifrelemedir. Bugün şifreleme çalışmaları oldukça ilerlemiş, bilgisayarlar oldukça gelişmiştir. Fakat bu durum saldırganlar için de geçerlidir. Hassas bilgiler bilinen ve test edilmiş şifreleme yöntemleri ile saklanmalıdır. Ayrıca daha önce kırılması uzun zaman alan algoritmalar günümüzde daha kısa zamanda çözülebilmektedir. Dolayısıyla uygulama içindeki algoritmalar zamanla gözden geçirilmeli ve güncellenmelidir.

3.2.6 Parametre Manipülasyonu

Dağıtık algoritmalar modüller arasında parametre gönderirler. Eğer bu parametreler arada değiştirilirse, saldırı gerçekleştirilmiş olur. 1 dolara satın alınan Ferrari bu duruma bir örnektir. Borcun belirlenmesi için web formu kullanan uygulama bu formdaki rakamın http proxy kullanılarak manipüle edilmesi sonucu değer 1 dolara olarak değiştirilmiştir.

3.2.7 Hata Yönetimi

Bazı teknolojiler hataları kullanarak hata yönetimi gerçekleştirmektedirler. Hatalar geliştiriciler ve sistem yöneticileri için uygulama ile ilgili birçok önemli bilgi ihtiva ettiği için çok önemlidirler. Bununla birlikte; geliştirici için bu derece önemli olan bilgi kullanıcı açısından problem oluşturabilmektedir. Her ne kadar kullanıcılar bu hataların ne demek olduğunu anlamasalar da saldırganlar için büyük ipuçları, yazılımla ilgili önemli bilgiler içermektedir. Bundan dolayı sadece genel bir hata mesajının dönmesi, hataların kayıt altında tutulması ve gerçek hataya sadece yöneticiler ulaşmasını sağlayacak sürecin oluşturulması gerekmektedir.

3.2.8 Kayıt Tutma ve Denetim

Uygulama veya uygulamanın yöneticileri saldırı altında olduklarını anlamalıdır. Bu durum aslında neyin normal neyin anormal olduğunun belirlenmesi ile sağlanır. Bir uygulamaya ilişkin normal süreç ve şablon tanımlanmalı ve bunu dışında bir olay olduğunda saldırı ihtimali ele alınmalıdır. Örneğin, normal senaryoda bir uygulamaya dakikada ortalama beş kişinin erişmesi beklenirken bu sayı bine ulaşıyorsa muhtemelen bir “Servis Dışı” bırakma atağı söz konusudur.

4 SONUÇ

Hem ülkemizde hem de dünyada bilişim teknolojilerine özellikle de yazılımlara olan bağımlılığımız artmakta ve buna bağlı olarak siber alandaki tehditler de giderek artmaktadır. Yazılımlarımız üzerindeki siber tehdidi doğru ölçebilmek, strateji geliştirebilmek ve geliştirmelerimizi bu yönde uygulayabilmek için öncelikle temeli düzgün oturtulmuş bir güvenli yazılım geliştirme sürecine ve güvenlik farkındalığı yüksek olan birimlere ihtiyaç vardır.

Bu kapsamda uluslararası kurumlar tarafından belirli standartlar hazırlanmış olup, yazılım geliştirme ve güvenli yazılım geliştirme süreçleri için kontrol noktaları oluşturulmuştur. Bunlardan en geçerli olanları; ISO/IEC 27001 Bilgi Teknolojileri- Güvenlik Teknikleri- Bilgi Güvenliği Yönetim Sistemi- Gereksinimler Standardı (ISO 27001), Yetenek Olgunluk Model Entegrasyonu (CMMI – Capability Maturity Model Integration) ve ISO/IEC 12207 Yazılım Yaşam Döngüsü Süreçleri Standardıdır.

Sonuç olarak; yazılım güvenliği, günümüzün yazılım bağımlı toplumunda önemli bir sorun haline gelmiştir. Güvenli uygulamalar çeşitli güvenli yazılım geliştirme yöntemleri kullanılarak elde edilir. Güvenli yazılım geliştirmek için güvenlik konusu yazılım geliştirme yaşam döngüsünün her aşamasında göz önünde bulundurulmalıdır. Yazılım geliştirme aşamaları güvenlik açıklarını minimize eden tek bir ortak amacı içermelidir. Güvenli yazılım geliştirme süreci kullanılmadan geliştirilen yazılımların oluşturduğu açıkların sonradan fark edilerek düzeltilmesi ve buna ait yamaların yayınlanmasının, şirketleri hem mali hem de itibar bakımından kayba uğrattığı gerçeği her zaman göz önünde bulundurulması gereken bir gerçektir.

Çalışma içerisinde son bölümde bahsetmiş olduğum yazılım geliştirme ve güvenli yazılım geliştirme süreçleri için kontrol noktaları oluşturan standartlara yer verilememiştir. Bu standartlar ile süreçlerin sentezlenmesi ileriki çalışmalara konu olmalıdır.

KAYNAKÇA

[1] Wikipedia - Waterfall model : < http://tr.wikipedia.org/wiki/Waterfall_model >

[2] PRO-G – Yazılım Geliştirme Yaşam Döngüsü ve Güvenlik: < http://www.pro-g.com.tr/pciveriguvenligi2008/pdf/pro-g-presentation-v1.1.pdf >

[3] Ulusal Bilgi Güvenliği Kapısı – Yazılım Geliştirme Süreçleri ve ISO 27001 Bilgi Güvenliği Yönetim Sistemi: < http://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/yazilim-gelistirme-surecleri-ve-iso-27001-bilgi-guvenligi-yonetim-sistemi.html >

[4] IDC – 2014 Worldwide Software Developer and ICT-Skilled Worker Estimates: < http://www.idc.com/getdoc.jsp?containerId=244709 >

[5] Ali Mersin – Güvenli Uygulama Geliştirme, Güvenli SDLC ve Kurumsal Deneyimler: < http://www.appsectr.org/presentation/appsectr_sdlc.pdf >

[6] Microsoft – Security Development Lifecycle: < https://www.microsoft.com/security/sdl/default.aspx?mstLocPickShow=True >

[7] $(ISC)^2$ – The Ten Best Practices for Secure Software Development: < https://www.isc2.org/uploadedFiles/(ISC)2_Public_Content/Certification_Programs/CSSLP/ISC2_WPIV.pdf >