

Açıklık Denetimlerinin Yazılım Güvenliđindeki Yeri

Sektörden bir arkadaşım bir keresinde şöyle söylemişti: “Güvenlik işi moda işi, biri ne yaparsa diğerleri de onu yapıyor”. Bu sözü sadece güvenliğe maletmek çok doğru olmaz, ancak güvenlik için bu durumu çok net gözlemliyoruz.

Yazılım güvenliği ile ilgili moda uygulama bitmiş bir uygulamanın açıklık denetimine tabi tutulmasıdır. Bu noktada açıklık denetiminin hakkını yememek, önemini küçümsememek, ancak tek başına açıklık denetimine dayanarak edinilen güvencenin yetersizlik ve verimsizliğini de net olarak ifade etmek istiyorum. Açıklık denetiminin hakkını yememeliyim, çünkü güvenli yazılım geliştirme yaşam döngülerinin önemli bileşenlerinden biridir, önemini küçümsememeliğim çünkü denetim yapan kişilerin ve kullanılan yöntemlerin yeterliliđi önemlidir ve denetimin sonuçlarının kalitesi üzerinde son derece etkilidir. Fakat tek başına verdiği güvence yetersizdir. Çünkü sonuçları ancak başınız çok ciddi belada ile başınız belada arasında yorumlanabilir. Güvenlik derecenin ölçümü konusundaki kesinliği son derece düşüktür. Ve yine fakat tek başına uygulandığında yazılım güvenliği açısından verimsizdir. Çünkü tüm yazılım mühendislerinin (ya da proje disiplini bilgisine sahip herkesin) bildiđi gibi yazılımdaki hataları düzeltmenin maliyeti yazılım geliştirme sürecinin başından sonuna doğru katlanarak artmaktadır. Güvenlik hataları için de bu durum geçerlidir. Açıklık denetimi bitmiş bir uygulama üzerinde yapıldığından olabilecek son noktada yapılan bir kontroldür. Buradaki kilit ifade “tek başına” açıklık denetiminin yetersizliğidir.

Bunlarla birlikte yapılan açıklık denetimlerinin zamanlaması ve şekline ilişkin bir eleştirim yoktur. Açıklık denetiminin tam da bugün yapıldığı şekilde, bitmiş uygulamanın gerçek ortam üzerinde yüklenmiş hali üzerinde yapılmalı, sadece yazılımdan kaynaklanan açıklıkların değil, üzerinde çalıştığı ortamdan kaynaklanan açıklıklarda bu şekilde tespit edilmesi gereklidir. Açıklık denetiminin (mecburen) tek başına uygulama alanı bulduğu bir konu satın alınan yazılımlar için satın alan kurumun yaptığı açıklık denetimidir. Esas itibariyle bu da yeterli değildir, ancak pratiklik ve yazılım üreticisi üzerinde oluşturacağı güvenlik hassasiyeti nedeniyle belli bir etkisi bulunmaktadır.

Çözüm

Eđer yazılım güvenliği konusunda ciddiysek, yani güvenlik düzeyini ölçebilecek kadar kontrolü ele alabilmek, yazılım kodunda bulunan açıklıkların yanı sıra yazılım mimarisinden kaynaklanan açıklıkları da bertaraf etmek istiyorsak yazılım geliştirme yaşam döngümüzü (hangi modeli benimsersek benimseyelim) güvenlik kontrollerini başa doğru öteleyerek düzenlemelisiniz. Bu literatürde “sola ötelemek” şeklinde ifade edilmektedir, çünkü soldan sağa akışta açıklık denetimi en sonda (yani en sağda) bulunmaktadır. Bu şekilde açıklıkların tasarım safhalarında bertaraf edilerek güvenlik maliyetinin de azaltılması mümkün olacaktır. Yazılım güvenliği konusundaki metodolojilerin detayına burada girmem mümkün değil ancak Gary McGraw tarafından geliştirilen “7 Touchpoints” ve Microsoft tarafından geliştirilen “Secure Development Lifecycle – SDL” metodolojilerini okumanızı öneririm. Özellikle Gary McGraw’un yazılım geliştirme yaşam döngülerinden bağımsız güvenlik kontrolleri (7 Touchpoints) her kurumun kendine adapte edebileceđi niteliktedir.

Bugün itibariyle ağ güvenliği gibi yazılım güvenliği de operasyon bölümlerine havale edilmiş durumdadır. Bugün uygulanan yazılım güvenlik yaklaşımı da bundan kaynaklanmaktadır. Yazılım güvenliğini ciddiye alan kurumların “Yazılım Güvenlik Mühendisleri” yetiştirmeleri ve güvenlik faaliyetlerini yazılım geliştirme ekiplerinin içine taşımaları gerekmektedir. Bu argüman daha gerçek anlamda bilgi güvenliği yöneticisi dahi bulunmayan kurumlarımız için biraz iddialıdır, ama unutmayalım Microsoft’un yazılım güvenliğine yaklaşımı Bill Gates’in 15 Ocak 2002’de yayınladığı memo ile ciddi bir değişime uğramıştır. Bu konu tüm dünya için yeni olmakla birlikte değişim kaçınılmazdır.

© 2009 BTRisk. Tüm hakları saklıdır.