Güvenli Yazılım Geliştirme Yaşam Süreci ve Statik Kod Analizi

Güncel SARIMAN, Ecir Uğur KÜÇÜKSİLLE

***Özet*— Geliştirilen yazılımların, teknolojik cihazlardaki kullanımı son zamanlarda giderek artmaktadır. Yazılımlar günümüz kritik altyapılarında, finansal sistemlerde, tıbbi bilgi sistemlerinde kullanılmaktadır. Sistemlerin çoğalmasıyla güvenli yazılımların geliştirilmesi, giderek artan güvenlik açıkları nedeniyle aynı oranda ilerleyememektedir. Yazılım güvenlik açıklarının artması ulusal alanda da ciddi bir tehdit oluşturmakta ve bu tehdit güvenli yazılım gelişimi için yeni yaklaşımları ortaya çıkarmaktadır. Yazılım ürünlerinde güvenilirliği arttırmayı düşünen kurumlar, yazılım güvenliğini keşfetmeye ihtiyaç duyarlar. Bununla birlikte güvenli yazılım kolay kolay gerçekleştirilememekte ve gerçek hayatta yatırımcılar yazılım geliştirme sürecinde yazılım ataklarına veya mevcut güvenlik açıklarına karşı direnememektedirler. Bu bildiride güvenli yazılım geliştirme modeli (G-YGYD) ve güvenliğin yazılıma olan etkisi anlatılmaktadır. Bildiride statik kod analizinin detaylarından ve güvenli yazılım geliştirme kodlama tekniklerine de değinilmektedir.**

***Anahtar Kelimeler*—Yazılım Güvenliği, Yazılım Yaşam Döngüsü, Statik Kod Analizi**

***Abstract*— The use of technological devices in developed software has been increasing in recent times. Software used in today's critical infrastructures, financial systems and medical systems. The development of secure software with the proliferation of systems cannot propagate at the same rate due to security vulnerabilities. Increase of Software vulnerabilities are a serious threat in the national area and this threat reveals new approaches for the development of secure software. Organizations, who minded to increase the reliability of the software products, need to explore software security. However, secure software cannot be accomplished easily and in real life, investors unable to resist existing vulnerabilities or software attacks in software development process. In this paper, secure software development model (S-SDM) and the impact of security to the software are described. Also details of static code analysis techniques and secure coding of software development are discussed.**

Keywords— Software Security, Software Life Cycle, Static Code Analysis

I. GİRİŞ

GÜNÜMÜZ teknolojisinde yazılım ürünlerine artan talep, yazılım geliştirme süreçlerinin derinlemesine araştırılmasını sağladı. Uygulama geliştirici firmalar kurumsal kimlik ve bilgileri korumak amacıyla bilgi güvenliği ve güvenlik mühendisliğinin önemini arttırmıştır [1]. CERT güvenlik raporuna göre yazılım güvenlik açıkları güvenlik uzmanlarıyla yakından ilgilidir [2]. Bilgi Güvenliği SSE-CMM[3], ISO/IEC 15408 [3] gibi standartlarla bağlantılıdır.

Güvenlik açıklarını engellemek için, güvenli yazılım geliştirme yaşam döngüsü ve mühendislik süreçlerindeki güvenlik gereksinimleri kapsamında birçok yazılım güvenlik aracı bulunmaktadır. Geleneksel olarak, yazılım güvenliği yazılım geliştirme yaşam döngüsünün(YGYD) başlarında dikkate alınmaz ve sadece geliştirmenin sonraki aşamalarında yazılıma entegre edilmeye çalışılır. Sonuç olarak, yazılım geliştirmenin çeşitli aşamalarında güvenlik açıklarının riskleri artmaktadır [8]. Güvenli yazılım mühendisliği YGYD 'nin başında ve ilerleyen aşamalarında güvenlik problemlerini dikkate alarak güvenlik açıklarını önlemeyi amaçlamaktadır. Güvenli yazılım mühendisliği tasarım, yapı ve test süreciyle güvenilirliği sağlamaktadır. Yazılım güvenliği, güvenli yazılım yaşam döngüsü süreçlerini ve güvenli yazılım geliştirme yöntemlerini içermektedir. Bir Güvenli Yazılım Geliştirme Yaşam Döngüsü (G-YGYD) süreci yazılım geliştirme yaşam döngüsü boyunca güvenli yazılım geliştirme yöntemlerini (GYG) kullanan güvenlik yöntemlerini dikkate alır. GYG yöntemleri, güvenlik özelliklerini, mühendislik süreçlerindeki güvenlik gereksinimlerini ve yazılım güvenliği güvence yöntemlerini içermektedir. Şunu belirtmek gerekir ki; yazılım güvenliği kaygıları uygulama güvenliği sorunlarından farklıdır. Uygulama güvenliği, yazılımın geliştirildikten ve yayınlandıktan sonra korunması ile ilgilidir. Genellikle giriş filtreleri, saldırı tespit sistemleri ve diğer koruma mekanizmalarını içermektedir[8]. Geliştirme yaşam döngüsü boyunca güvenlik konularına yol göstermesi açısından uygun G-YGYD süreci seçmek önemlidir. Ayrıca, eğer geliştirilen uygulamanın piyasaya sürümünden hemen sonra hata siliniyorsa, bir hatayı silmek için geliştirilen ve yayınlanan bir yama 200 kat daha fazla pahalı olabilmektedir. Hatalar göz önüne alındığında, uygun güvenli gereksinim mühendisliği yöntemlerinin kullanılması gereklidir.

Geliştirmenin en başından beri yazılımda güvenliği fikir olarak gören, çeşitli GYG yöntemleri önerilmiştir. Bununla birlikte, uygulamalar halen geleneksel YGYD modeliyle geliştirilmekte ve geliştiricilerin uygun gördüğü çeşitli GYG yöntemleri geliştirmenin farklı aşamalarda kullanılmaktadır. Geliştirme süresince farklı GYG yöntemlerini kullanan birçok G-YGYD süreci önerilmektedir. Bu G-YGYD süreçlerin her birinin kendi güçlü ve zayıf yönleri vardır [7].

Yazılım geliştirme sürecinde güvenlik açıklarının en sık rastlandığı bölüm olan statik analiz, yazılımı ve sistemlerinin

Manuscript received July 15, 2013. (Write the date on which you submitted your paper for review.)

G. SARIMAN Süleyman Demirel University, Electronics and Communicating Engineering, Isparta 32100 Turkey (corresponding author to provide phone: 505-804-8988; e-mail: guncelsariman@mu.edu.tr).

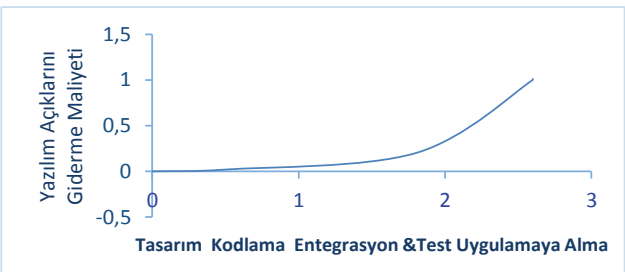
E. U. KÜÇÜKSİLLE, was with Süleyman Demirel University,Computer Engineering, Isparta, Turkey. (e-mail: ecirkucuksille@sdu.edu.tr).

güvenliğini geliştirmek için umut verici bir teknoloji olarak ortaya çıkmıştır. Statik analiz araçları, derleyiciler, manuel kod inceleme ve test etmek gibi geleneksel tekniklerle tespit edilemeyen hataları bulmak için yazılımları analiz eder. Bununla birlikte bir dizi kısıtlamalar yazılımın gelişimini engellemiştir [4].

Giderek artan yazılım projelerinin mimarisi ilk günkü gibi olmamakla birlikte işlevsellik gün geçtikçe artmakta ve beraberinde karmaşık ve detaylı bir projeyi oluşturmaktadır. Yazılımdaki kalite ve güvenliğin kaybıyla ilgili birçok problem, etkili bir şekilde yönetilemeyen karmaşıklığın büyümesi ile ilişkilendirilebilir [5]. Örneğin masaüstü işletim sistemlerinin kod tabanı şaşırtıcı bir şekilde artmaktadır. Microsoft işletim sistemi 1993 yılında 6 milyon, 2000 yılında 29 milyon kod satırından oluşurken, 2005 yılında 50 milyon kod satırına ulaşmıştır. Bir debian linux işletim sistemine ait kod satırı 2000 yılında 55 milyon kod satırından 2005 yılında 215 milyon kod satırına daha hızlı bir şekilde ulaşmıştır [6]. Güvenlik açıklarının sıklığı, yazılım karmaşıklığının etkisini izlemek için bir lider gibi davranır. CERT istatistiklerine göre, belgelendirilen güvenlik açıklarının sayısı katlanarak artıyor. Bu güvenlik açıkların sayısı 1999 yılında yaklaşık 400, 2002 yılında 4000'den fazla ve 2006 yılında 8000'den fazla olmuştur [2].

II. GÜVENLİ YAZILIM GELİŞTİRME

Yazılım, bir problemi çözmek amacıyla farklı cihazların birbirleriyle iletişim kurabilmesini sağlayan ve görevlerini ya da kullanılabilirliklerini geliştirmeye yarayan, bilgisayar dili kullanılarak oluşturulmuş bir ifadedir [10]. 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 [11]. 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 [7]. Şekil-1 de yazılım geliştirme sürecinde, yazılım açıklarının maliyet analizi gösterilmektedir.



Şekil. 1. Güvenli Yazılım Sürecinde Hata Düzeltme Maliyeti

III. YAZILIM GELİŞTİRME SÜRECİNDE GÜVENLİK UYGULAMALARI

G-YGYD güvenli yazılım geliştirme yaşam döngüsünün kısaltılmış halidir. Kısaca YGYD süreç, yöntem ve bir dizi metodolojiyi yazılım projeleri oluşturmak veya geliştirmek için uygular. Bu metodolojiler yeni bir yazılım modülü geliştirmek için tek bir yol tanımlar. Yazılım geliştirme yaşam döngüsünün temel ayağı olan 4 genel yaklaşım bulunmaktadır. YGYD metodolojisi şelale modeli, artırımlı model, spiral model, prototip modeli olarak sınıflandırılmaktadır.

Dizayn ve tasarım hataları, üzerinde çok düşünülmemiş güvenlik açıkları, yeni geliştirilen malware uygulamaları ve bunlar gibi birçok neden, geliştirilen uygulamaların debug ve patching süresini maksimuma çıkartır. Bu debug seansları uzun zaman almasının yanında yazılımcıyı moral olarak da kötü etkiler. Dolayısıyla programın yapımına harcanılan bütçeyi bile geçebilecek yatırımlar yapılması zorunlu hale gelir. Daha güvenli bir yazılım geliştirebilmek için güvenliğin yazılım yaşam döngüsüne dâhil edilmesi gerekmektedir. Genel olarak güvenlik YGYD sürecine 6 aşamada eklenebilmektedir [9]. Bu süreçler içerisinde sırasıyla eğitim, gereksinim analizi, tasarım, geliştirme, test ve dağıtım sıralanmaktadır. Şekil-2 de YGYD aşamaları Şekil-3 de ise G-YGYD aşamaları gösterilmektedir.

Güvenliği sağlamak için, güvenli geliştirme kavramlarını var olan geliştirme sürecine entegre etmek gerekmektedir. Güvenli yazılım geliştirme süreci yaşam döngüsündeki aşamalar aşağıda açıklanmıştır.

A. Uygulamalar İçin Temel Güvenlik Eğitimi

Bir yazılım geliştirme takımındaki geliştiriciler, test elemanları ve program yöneticileri gibi çalışanlar, güvenlik temelleri ve güvenlikteki son yaklaşımlar hakkında bilgili kalmak için uygun bir eğitim almalıdır. Güvenlik eğitimi aşağıdaki temel kavramları içermelidir;

a) -Güvenlik tasarımı

Saldırı yüzeyi azaltma, Derinlemesine savunma, En az ayrıcalık ilkesi

b) -Tehdit Modelleme

Tehdit modellemeye genel bakış, Bir tehdit modelin tasarım ilkeleri, Tehdit modelindeki kısıtları kodlama

c) -Güvenli kodlama

Arabellek taşmaları, Tamsayı aritmetik hatalar, Cross Site Scripting, Sql Injection, Zayıf Şifreleme

d) -Güvenlik Testi

Güvenlik testi ve fonksiyonel test arasındaki farklar, Risk değerlendirmesi, Güvenlik test yöntemleri

e) -Gizlilik

Gizlilik duyarlı veri türleri, gizlilik tasarımının en iyi uygulamaları, Risk değerlendirmesi, En iyi gizlilik geliştirme uygulamaları

Şekil. 2. Yazılım Geliştirme Yaşam Döngüsü (YGYD)

Şekil. 3. Güvenli Yazılım Geliştirme Yaşam Döngüsü (G-YGYD)

IV. GÜVENLİ YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ

A. Gereksinim ve Risk Analizi

Gereksinim aşaması, geliştirme takımlarının güvenlik, plan ve programın herhangi bir kesintiye uğramasını engellemeyi amaçlar. Güvenlik ve gizlilik ihtiyaç analizi proje başlangıcında yapılır ve uygulama için minimum güvenlik gereksinimlerinin özelliklerini içerir. Yazılım geliştirme yaşam döngüsü gereksinim aşaması ile başlar. Gereksinim aşamasının amacı müşterinin taleplerini yerine getiren yazılım gereksinimleri belirlemek ve aynı zamanda açık ve anlaşılır bir şekilde temsil etmektir. Yazılım ihtiyaçlarının belirlenmesi ile birlikte, çalışmalar güvenlik gereksinimlerini belirlemek için yapılır. Güvenlik gereksinimleri farklı kaynaklardan farklı zamanlarda gelmektedir. Bazı güvenlik gereksinimleri, tasarım aşamasında tehdit modellemeden sonra tanımlanır. Tanımlama yapıldıktan sonra, güvenlik gereksinimleri diğer yazılım gereksinimleriyle birlikte analize tabi tutulur. Analizden sonra, gereksinimler belgelendirilir. Gereksinim belgeleri işlevsel, işlevsel olmayan ve güvenlik gereksinimlerini içermektedir. Bu belgeyi kullanarak, proje takımı Use Case diyagramları oluşturur. Use Case ler yüksek düzeydeki sistemler için önemli bir tekniktir. Yeni gereksinimler tanımlandığında var olan gereksinimlerin değiştirilmesi muhtemeldir ve bazı gereksinimler use case diyagramları yanlış kullanım durumlarında silinmektedir. Bu gereksinimler, gereksinim dokümanlarını güncellemek için çağrıda bulunur.

B. Tasarım ve Tehdit Modelleme

Tasarım özellikleri, kullanıcının doğrudan maruz kalacağı güvenlik ve gizlilik esaslarını açıklamalıdır. Ayrıca, fonksiyonların güvenli bir şekilde nasıl uygulanacağını da açıklar. Tasarım aşaması boyunca güvenlik ve gizlilik kaygılarını dikkate almak çok önemlidir. Tasarım aşaması yazılım mimarisinin oluşturulması ile başlar. 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 bir bilgi verir. Güvenlik tasarımının bir sonraki aşaması ise varlık ve erişim noktalarını belirlemektir. Bu varlıkların erişim noktaları da daha önceden tespit edilen erişim noktalarından olabilir. Güvenlik tasarımındaki diğer bir nokta ise saldırı yüzeyini en aza indirgemektir. Bu aşama çok çaba gerektirmemekle beraber, saldırganın çalışma alanını azaltmanın en etkili yoludur. Yazılım tasarımı ayrıntılı olarak analiz edilir ve maximum kullanıcı sayısının ihtiyaçlarını etkilemeden etki altında kalan yüzey alanının azaltılması konusunda çaba gösterilir. Tasarımdaki diğer bir nokta ise yazılım için geçerli olan tehditleri tespit etmektir. Tehdit modelleme güvenli yazılım aşamasında önemli bir kilometre taşı olarak kabul edilebilir. Tehdit modelleme güvenlik sorunlarının bileşen veya uygulama seviyesinde dikkate alınmasını sağlar [12]. Güvenlik tasarım 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, güvenlik tasarımı şirketler için devamlı olarak güncellenir [13].

C. Uygulama

Güvenlik perspektifinden bakıldığında uygulama aşaması iki aşamalı rol oynamaktadır. İlk rol yazılıma giren güvenlik açıklarını ve ikinci rol ise yazlımda mevcut olan açıkları engellemektir. İlk rol güvenli kod yazarak gerçekleştirilir. Güvenli kod yazmak oldukça önemlidir, çünkü yapılmadığı takdirde bütün çabalar boşa gidecektir. Güvenli kod, tasarım aşamasında(tasarım aşamasında kurulan güvenlik tasarımı) 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 rol ise güvenlik açıklarını tespit etmek için statik analizin kullanılmasıdır. Geliştirme ekibi tarafından yazılmış kod otomatik araçlar yardımı ile analiz edilir. Bu otomatik araçlar ortak güvenlik açıklarını arar [13].

Geliştirme ekipleri yeni güvenlik analizi ve korumalarından yararlanmak için onaylanmış araçların en son sürümünü kullanmalıdır. Bu araçların listesi proje ekibi için güvenlik danışmanı tarafından onaylanmalıdır[12]. Ayrıca proje ekibi kullanılan yazılımın api ve fonksiyonlarını kontrol etmelidir çünkü kullanılan birçok api ve fonksiyon güncel saldırı ve tehditler karşısında güvenli değildir. Bu fonksiyonlar güvenli olanlarıyla değiştirilmelidir. Otomatik statik analizden sonra gözden geçirmeler yapılmaktadır. Gözden geçirme yazılım kodundaki güvenlik açıklarını bulmanın etkili yoludur.

D. Üretim ve Test Aşaması

Geliştirilen uygulama yayınlanmadan önce doğrulama ve

test aşamalarından geçirilmelidir. Bu aşamalar ise dinamik kod analizi, fuzz testi, tehdit modeli ve saldırı yüzey incelemesidir. Bir uygulamadaki zayıflıkları tespit edebilmek için uygulamaya rastgele veriler göndererek, uygulamanın farklı isteklere verdiği cevaplar analiz edilir. Bu rastgele veri gönderme işlemine fuzzing (fuzz testing), bu işlemi yapmaya yarayan araçlara ise fuzzer denilmektedir. Fuzz testing; test yapılacak uygulamaya ait işlevsel ve yazılımsal özellikler bilinmediğinden ve gönderilen verilerin uygulamada oluşturduğu etki ve sonuçların analizine dayandığından, uygulama test tekniklerinden black-box kategorisine girmektedir. Fuzz testing; yazılım geliştirme, test süreçlerinde ve ters mühendislik işlemlerinde kullanıldığı gibi, web uygulamalarında zayıflık tespiti veya bilgi toplama amaçlı da kullanılmaktadır [14].

Test aşaması test planlamasıyla başlar. Güvenlik test durumları, yazılımları güvenlik seviyesinde başarılı bir şekilde savunmaları için tasarlanmıştır. Test planlaması boyunca, yazılım tasarımındaki herhangi bir değişiklik test ekibine iletilmelidir aksi takdirde test takımı değişiklikleri farketmeden analize devam edecektir. Yazılımın doğasına dayanarak uygulanabilir tehditler, hata şiddeti gözden geçirilir çünkü yazılım kodundaki küçük bir hatayı düzeltmek birkaç dakikayı alabilir fakat güvenliğe olan etkisi büyük olabilir. Test planlamasından sonra bir test sistemi tasarlanır. Test sistemi, muhtemelen test doğasına bağlı olarak çeşitli yazılım ve donanım ortamları içerir. Test süresince geliştirme takımı ve test takımı iş birliği halindedir. Test takımı güvenlik açıklarını tanımlayarak geliştirme ekibine raporlar ve geliştirme ekibi açıkları giderir.

E. Yayın ve Dağıtım

Yazılımın geliştirilmesi bittiğinde, son kullanıcıya dağıtıma başlanmadan hemen önce geliştirilen yazılımın güvenli olup olmadığının kontrol edilmesi ve her şeyinin güvenli olduğundan emin olunması gerekmektedir [15]. Gözden geçirme yapılarak kalan güvenlik açıklarının gözden geçirilmesi gerekmektedir. İnceleme sonunda gözden geçirme raporu üretilir. Geliştirme ekibi raporda tanımlanan güvenlik açıklarına karşı kodu düzeltir. İncelemeden sonra, bir güvenlik denetimi yapılır. Tüm proje ekibi denetimde yer almaktadır. Eğer sorunlar denetim sırasında tespit edilirse, yazılımda değişikliğe gidebilir. Denetim sonunda denetim raporu oluşturulur. Denetim raporuna göre, proje yönetimi yazılımın yayınlanma durumuna karar verir. Yayınlandıktan sonra yazılım dağıtım için hazır hale gelir.

V.GÜVENLİ YAZILIM GELİŞTİRME SÜRECİNDE STATİK KOD ANALİZİ

Güvenli yazılım geliştirme süreci, geliştirilen projeye güvenliğin yanı sıra mali açıdan önemli katkılar sağlamaktadır. Yazılımda ortaya çıkan açıklara yama geliştirmek, bu açıkları daha yazılım geliştirme sürecinde fark edip önlemeye nazaran çok daha fazla mali yük getirmektedir. Üstelik sonradan ortaya çıkacak olan bu güvenlik açıkları şirket imajının zedelenmesi, hukuki açıdan sorunlar, maddi kayıplar, şirketin borsa hisse değerinin düşmesi ve hatta şirketin iş hayatının son bulması gibi sonuçlar dahi doğurabilirler [16]. G-YGYD süreciyle beraber tehdit modelleme, risk analizi, yazılımcılar için güvenlik eğitimleri ve dokümanları, mimari analiz, kaynak kod analizi, sızma testleri gibi güvenlik için vazgeçilmez gereksinimler bir bütün olarak yazılım sürecine entegre edilirler. Statik kod analizi de bu süreçte önemli bir rol oynamakta ve yazılımın çalıştırılmadan kodlama zamanında analiz edilmesini ve olası hataların tespit edilmesini sağlamaktadır.

Yazılımın koşturulması sırasında fark edilemeyecek veya geç fark edilecek bazı hatalar statik kod analizleri ile bulunabilir. Statik kod analizleri yazılımın kalitesi, çalışma performansı, işlevselliği, güvenirliliği açılarından önemli olmakla beraber statik kod analizlerinin temel mantığı kod üzerinde sorgulamalar gerçekleştirmektir. Gözden geçirme yoluyla da yapılabilen statik kod analizi, günümüzde otomatik kod analiz araçlarıyla zamansal ve ekonomik maliyet açısından daha uygun olmasından dolayı hızlandırılmıştır. Geliştirilen uygulamaların üzerinde gerçekleştirilen hata tespiti vb. durumların yazılım yaşam döngüsünün ilerleyen aşamalarında giderek artmasından dolayı yazılımın erken aşamalarında kullanılan otomatik statik kod analiz araçları, daha kaliteli ve maliyetli uygun yazılımların geliştirilmesine olanak sağlamaktadır. Farklı boyutlardaki yazılımlara ait hatalar benzer maliyetlerle tespit edilebilmektedir. Statik analiz araçları daha çok atama ve denetim gibi programlama hatalarını tespit ederken, gözden geçirme ile yapılan kod analizi fonksiyonel hataları da ortaya çıkarabilmektedir[17]. Gözden geçirmeyle yapılan aşamada dikkat edilmesi gereken nokta ise hataların gözden kaçırılabilir olmasıdır.

A. Statik Kod Analizinde Kullanılan Yöntemler

Geliştirilen yazılımlardaki kodlar çeşitli kategorilerde incelenebilmektedir. Statik analiz; duyarlı fonksiyon analizi, veri akış analizi, fonksiyon değer analizi, kodlama standartı analizi, bilgi akış analizi ve yanlış yol budama analizi olarak kategorize edilebilmektedirler.

Duyarlı fonksiyon analizi ile fonksiyon/prosedürler-arası ilişkiler ve veri akışları sorgulamaları gerçekleştirilir. Fonksiyon değer analizi, yazılımdaki fonksiyonların dönüş değerlerinin analizinin yapıldığı yöntemdir.

Veri Akış analizinin amacı ise değer atanmadan bir değişkene erişebilen verilerin yazılımda uygulama yollarının olmadığını göstermektir.

Bilgi akış analizi, bir birim kodun yürütülmesi sırasında koddaki giriş ve çıkışlar arasında nasıl bir bağımlılık kurulacağını tanımlar. Bu analiz, yazılım/donanım ara yüzündeki girişlere doğru tüm yolları izleyebilen kritik çıkışlar için önemlidir.

Yol fonksiyon analizi, resmi bir belirtim gerektirmeden, matematiksel işlemler vasıtasıyla bir programın özelliklerini doğrulamak için kullanılır. Çok sayıda yolun bulunduğu durumlarda, bazı yollar hiç çalıştırılmadan kalabilir. Yazılım içinde çalışması mümkün olmayan yollar belirlenir.

Kodlama Standartı analizinde, çalışan kod yazmanın yanında iyi kod yazmanın gerekliliği vurgulanmaktadır. Kodlama yaparken isimlendirme standartlarına dikkat edilmelidir. Sık kullanılan isimlendirme standartları şu şekildedir;

Kod geliştirilirken ara yüz isimlendirmelerinde "I" ön eki, değişkenler için anlamlı ve açıklayıcı kelimeler

kullanılmalıdır. Kısaltmalar kullanılmamalıdır. Tek karakterlik değişken isimleri kullanılmalıdır, fakat döngülerdeki parametreler tek karakterlik olabilmektedir. UI elemanlarını kullanırken programlama dilinin belirlediği kısaltmalar kullanılmalıdır. Sınıfların bulunduğu fiziksel dosya isimleri sınıf ismi ile aynı olmalıdır. Editörde kod geliştirirken daha anlaşılır kod satırları için girintiler ve yorum satırları kullanılmalıdır. Kullanılan süslü parantezler if, for gibi kodlarla aynı satırda olmalıdır. Kod parçalarını gruplamak için varsa editöre ait gerekli kod kullanılmalıdır. Yazılımdaki hata ayıklama türleri detaylandırılmalı ve ne tür hata alınabileceğine göre detaylandırılmalıdır. Aynı zamanda fonksiyonlara isim verirken kısaltmalardan kaçınılmalıdır ve yorum satırına gerek olmadan anlamlı isimler verilmelidir.

B. Statik Kod Analiz Araçları

Birçok kaynak kod analiz programı aynı kaynak dosyada olmayan kodlar arasında karmaşık etkileşimlerin sebep olduğu hataları bulan tam bir program analizi gerçekleştirir. Araçların bir kısmı yazılımın kaynak kodunu metinsel olarak incelemektedir. Daha kapsamlı analiz yapan araçlar kullanılan programlama diline ve editöre ihtiyaç duymaktadır. Microsoft .Net ortamında FxCop, Gendarme, StyleCop, Java platformunda CheckStyle, Findbugs, Hammurapi, PMD araçları geliştiriciler tarafından sıkça kullanılan analiz araçlarıdır.

VI. SONUÇ

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 farkedilerek düzeltilmesi ve buna ait yamaların yayınlanmasının, şirketleri hem mali hem de prestij bakımından kayba uğrattığı gerçeği her zaman göz önünde bulundurulması gereken bir gerçektir.

KAYNAKLAR

[1] F. J. B. Nunes, D. A. Belchior, A. B. Albuquerque, A Knowledge Management Approach to Support a Secure Software Development, International Conference on Availability, Reliability and Security, March. 16-19, 2009, s. 829-834

[2] CERT, Coordination Center Statistics. www.cert.org/stats/cert_stats.html.

[3] ISO/IEC 15408-1, Information technology – Security techniques – Evaluation criteria for IT security, 2005.

[4] N. D. Kleidermacher, "Integrating Static Analysis into a Secure Software Development Process Technologies for Homeland Security", 2008 IEEE Conference, May. 12-13, 2008, s. 367 – 371.

[5] Robert Parker, Are Vendors Doing Enough To Improve Software?. Optimize Magazine; http://www.optimizemag.com/issue/009/squareoff.htm.

[6] Wikipedia Source lines of code http://en.wikipedia.org/wiki/Source_lines_of_code.

[7] M. U. A. Khan, M. Zulkernine, "On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software" Computer Software and Applications Conference. July. 20-24, 2009, s. 353–358.

[8] G. McGraw, Software Security: Building Security In, Addison Wesley, 2006.

[9] G. İ. Özbilgin, 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

[10] Yazılım, Türk Dil Kurumu Büyük Sözlük, http://tdkterim.gov.tr/bts/

[11] B. Dayıoğlu, Yazılım Geliştirme Yaşam Döngüsü ve Güvenlik, http://www.pro g.com.tr/ pciveriguvenligi 2008/pdf/pro-g-presentation-v1.1.pdf

[12] New SDLC: Security Development Life Cycle. http://codespread.com/new-sdlc-security-development-life-cycle.html.

[13] S. R. Ahmed, Secure Software Development-Identification of Security Activities and Their Integration in Software Development Lifecycle. School of Engineering Blekinge Institute of Technology.Yüksek Lisans Tezi. 2007.

[14] O. Yilmaz, Fuzz Testing. http://webguvenligi.org/dergi/FuzzTesting-Agustos2009-OnurYilmaz.pdf.

[15] E. Burak, Microsoft'un Güvenli Geliştirme Süreci (SDL). http://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/microsoftun-guvenli-gelistirme-sureci-sdl.html.

[16] İ. E. Tatli, Java'da Güvenli Yazılım Geliştirme. www.webguvenligi.org/.../Javada_Guvenli_Yazilim_Gelistirme_OWASP

[17] Secure Coding Guidelines for the Java Programming Language, http://java.sun.com/security/seccodeguide.html

Güncel SARIMAN 1986 yılında Muğla'da doğdu. Süleyman Demirel Üniversitesi Teknik Eğitim Fakültesi Bilgisayar Sistemleri Öğretmenliği lisans eğitimini tamamladı. Yüksek lisans eğitimini Süleyman Demirel Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Ana Bilim Dalında yaptı. Doktora eğitimi Süleyman Demirel Üniversitesi Mühendislik Fakültesi Elektronik ve Haberleşme Mühendisliği Ana Bilim Dalında tez aşamasında devam etmektedir. 2010 ve 2012 yılları arasında Süleyman Demirel Üniversitesi Bilgi İşlem Daire Başkanlığı yazılım biriminde yazılım mühendisi olarak çalışmıştır. Halen Muğla Sıtkı Koçman Üniversitesi Bilgi İşlem Daire Başkanlığı yazılım biriminde yazılım mühendisi olarak çalışmaktadır. Bilgisayar, güvenlik ve yapay zeka alanlarında çalışmaları bulunmaktadır.

Ecir Uğur KÜÇÜKSİLLE 1976 yılında Isparta'da doğdu. Gazi Üniversitesi Teknik Eğitim Fakültesi Bilgisayar Sistemleri Öğretmenliği lisans eğitimini tamamladı. Yüksek lisans eğitimini Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Makine Eğitimi Ana Bilim Dalında yaptı. Doktora eğitimini Süleyman Demirel Üniversitesi Sosyal Bilimler Enstitüsü İşletme/Sayısal Yöntemler Ana Bilim Dalında tamamladı. Halen Süleyman Demirel Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde öğretim üyesi olarak görev yapmaktadır.. Bilgisayar, güvenlik ve yapay zeka alanlarında çalışmaları bulunmaktadır.