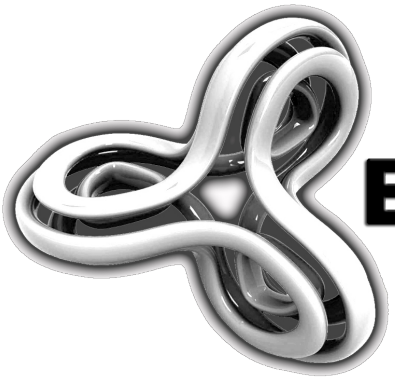
2015



BGA

BİLGİ GÜVENLİĞİ AKADEMİSİ

www.bga.com.tr

[GÜVENLİ YAZILIM GELİŞTİRME EĞİTİMİ]

[Bu doküman, güvenli yazılım geliştirme eğitimi içeriğini oluşturma amacı ile hazırlanmış olup eğitim temel başlıklarını içermektedir.]

İÇERİK

1. Giriş 4
1.1 Tanışma ve İçeriğin Detaylandırılması 4
1.2 Doğrulama Eğilimi (Confirmation Bias) 4
1.3 Yazılım Güvenliğine Genel Bakış 4
1.4 Altyapı - Web Standartları 5
2. Lab Ortamı 5
3. Güvenli Girdi Kontrolü 5
3.1 Cross Site Scripting 5
3.2 SQL İnjection 6
3.3 Diğer Yetersiz Girdi Kontrolü Zafiyetleri 6
3.4 Girdi Kontrolü Stratejileri 7
4. Güvenli Kimlik Doğrulama 9
4.1 Kimlik Doğrulama Çeşitleri 9
4.2 Sözlük/Kaba Kuvvet Kontrolü 9
4.3 Kimlik Doğrulama Stratejileri 10
5. Güvenli Oturum Yönetimi 10
5.1 Önceden Belirlenmiş Oturum (Session Fixation) 10
5.2 Cross Site Request Forgery (CSRF) 10
5.3 Oturum Uygulama Zayıflıkları (Session Puzzling/Overriding) 10
5.4 Oturum Korsanlığı 11
5.5 Oturum Yönetimi Stratejileri 11
6. Güvenli Yetkilendirme 11
6.1 Directory Traversal 11
6.2 Güvensiz Direk Nesne Referansı 11
6.3 Açık Yönlendirmeler (Open Redirects) 11
6.4 Forceful Browsing 11
6.5 Yetkilendirme Stratejileri 12
7. Güvenli Dizayn 12
7.1 Güvensiz İş Mantığı Kontrolleri 12
7.2 Tehdit Modelleme 12
8. Güvenli Hata Yönetimi ve Kayıt Tutma 13
8.1 Yetersiz Hata Yönetimi 13
8.2 Try/Catch/Finally Blokları 13
8.3 Kayıt Sahteciliği 13
8.4 Güvenli Hata Yönetimi ve Kayıt Tutma Stratejileri 13
9. Güvenli Kriptografi 14

9.1 Kriptoğrafi Giriş ve Güvenli Algoritmalar 14
9.2 OWASP ESAPI Java - Encryptor / Randomizer 14
9.3 .NET Çatısı - Security.Cryptography 14
10. Statik Kaynak Kod Analizi 14
10.1 Statik Kaynak Kod Analizi Temelleri 14
10.2 Statik Kaynak Kod Analiz Araçları 15
11. Güvenli Web Servisleri ve AJAX 15
11.1 Web Servisleri Zafiyetleri ve Önlemleri 15
11.2 Güvenli Web Servisleri Stratejileri 15
11.3 AJAX Zafiyetleri ve Önlemleri 15
11.4 Güvenli AJAX Stratejileri 16

1. Giriş

Güvenli yazılım geliştirme eğitimi, daha çok web uygulamalar ağırlıklı zafiyetlerin önlenmesi için gerekli yapıların kullanılması ve bazı durumlarda üretilmesini anlatmaktadır. Eğitim, güvenli yazılım geliştirme stratejilerini ayrı kategoriler içinde özelleştirir ve detaylandırır.

Güvenli yazılım geliştirme eğitimi, Java ve .NET dünyasından örnekler barındırmaktadır. Anlatılan konular lab ortamında koda dokunularak öğrenilir ve önlemler gerçekleştirilir.

1.1 Tanışma ve İçeriğin Detaylandırılması

Tanışma ve Beklentilerin Alınması

Eğitimin bu bölümünde katılımcılar kısaca kendilerini, işlerini anlatırlar ve eğitimden beklentilerini sıralarlar.

Geliştirici Perspektifinden Güvenlik

Saldırgan ve geliştirici arasında temel farklar vardır. Geliştirici bir uygulamanın nasıl çalışması gerektiğini hedef alan bir yaklaşım içerisindedir. Gereksinimler ve kullanım senaryoları (use case) dokümanlarını baz alarak dizaynlarını geliştirirler. Buna karşın saldırgan, bir uygulamanın ne için çalıştığını ve çalışma prensibindeki atlatma durumlarını hedef alır.

Önlem, sistem yöneticileri veya geliştiricilerin sistemlerinde veya kodlarında bulunan açıklıkları eksiksiz kapatabilmeleridir. Zaman, bulunan açıklıkların önemine göre kısıtlıdır. Bu bakış açısında, çalışan teknolojilerin nasıl çalıştıklarının derinlemesine bilinmesi başka bir zafiyet oluşturmamak veya zafiyetleri tamamen kapatabilmek adına çok önemlidir.

Eğitim, güvenli geliştirme için prensipler ve kontrol bakış açısına sahiptir ama yazılımlarda bulunabilecek zafiyetlerin nasıl kötüye kullanılabilecekleri ("hacking") hakkında detaylar da içerecektir.

Eğitim İçeriği ve Takvimi

Eğitim süresinde verilecek konuların başlıkları ve kısa kısa içerikleri gösterilecektir. Eğitim günleri, günlük kaç saat süreceği ve hangi konuların hangi saatlerde verileceği gösterilecektir.

1.2 Doğrulama Eğilimi (Confirmation Bias)

Güvenli yazılım geliştirme sürecinde geliştiricilerin en büyük kozu kaynak kodun kendileri tarafından üretilmiş olması, iş mantığında kullanılan algoritmaların büyük bölümünün kendileri tarafından geliştirilmiş olmasıdır. Bu bilgilerin yanında bir geliştiricinin güvenli kod üretmesi için sahip olması gereken en kritik davranış biçimi "Şüpheci Yaklaşım", diğer bir deyişle "Doğrulama Eğilimi"nin tersine olgusudur. Bir Billişsel Bilim (Cognitive Science) terimi olan bu kavram, katılımcıların uygulamalarını geliştirirken sahip olmaları gereken bakış açısına vurgu yapacaktır.

1.3 Yazılım Güvenliğine Genel Bakış

Güvenliğin yazılımlar için yeni bir kavram gibi gözükmesi yanlış bir algının sonucudur. Yazılımların başlangıç zamanlarından beri önem arz eden bu konu, İnternet'in gelişmesi ve web uygulamalarının ve dolayısıyla yazılımların üzerindeki kötü amaçlı gözlerin artması ile kendini iyiden iyiye belirgin hale getirmiştir.

Yazılımda Güvenlik Yöntemleri ve Modelleri

Son yıllarda, yazılım için her konuda olduğu gibi sistematik bir anlayışın gerekliliğine paralel olarak, güvenlik için de metodik yöntemler ve olgunluk modelleri geliştirilmiş ve uygulanmaya konmuştur. Bu bölümde, tehdit modelleri, saldırı ağaçları, yazılım olgunluk modelleri konularına değinilecektir.

Bir saldırganı geliştiriciden ayıran en önemli konu, uygulamalara yaklaşımıdır. Geliştirici uygulama gereksinimlerinden ve kullanım senaryolarından yola çıkarak uygulamayı geliştirirken, saldırgan uygulamanın akışlarını atlatmaya çalışmaktadır. Bu nedenle güvenlik anlamında uygulama gereksinimlerine ek olarak güvenlik gereksinimleri ve kullanım senaryolarına ek olarak kötü kullanım senaryoları geliştirmenin erken safhalarında üretilmelidir. Bu iki ek yöntem ile yazılımın güvenlik gereksinim seviyesi hesaplanır ve çok detaylı olmayan kötü kullanım senaryoları listelenmiş olur.

1.4 Altyapı - Web Standartları

Web standartların bilinmesi yazılım güvenliğinin en önemli parçalarından biri olan web güvenliği konularının anlaşılması için kritik önem taşımaktadır.

HTML/Javascript/DHTML

Web'in üzerinde durduğu HTML, Javascript ve DOM standartları incelenecek ve temel örnekler verilecektir. Özellikle DHTML olarak adlandırılan DOM ve Javascript, istemci taraflı zafiyetlerde kullanılmaktadırlar. Kodlama tekniklerinden HTML Kodlama (HTML Encoding) konusu da bu bölümde incelenecektir.

HTTP/URL

Web iletişim protokolü olan HTTP, HTTP istek/cevap çeşitleri ve bölümlerinin anlaşılması denetimlerin gerçekleştirilmesinde büyük önem taşımaktadır. Web adresleme sistem olan URL zafiyetlerin gerçeklenmesinde en önemli parçalardan biridir. Özellikle URL kısımları (Query, Anchor, Domain, v.b.) ve URL kodlama (URL Encoding) konuları bu bölümde incelenecektir.

SQL

Bu bölümde, uygulamaların arka uç tarafında verileri saklamak ve göstermek için kullandıkları en yaygın teknoloji olan veritabanlarının sorgulama dili olan SQL ana ilkeleri ile anlatılacaktır. Ayrıca binary search, ascii tabloları gibi temel ve önemli olgulara da değinilecektir. Bu şekilde en tehlikeli güvenlik açıklarından biri olan SQL İnjection zafiyetinin anlaşılması için temel oluşturulacaktır.

Web Uygulama Kavramları

İstemci taraflı standartların yanında web uygulamaların çalışma mantığı da denetimlerin kalitesini ve etkisini doğru orantıda arttıracaktır. Web uygulamalarına genel bakış, cookie'ler ile oturum yönetimi, kritik aynı kaynak politikası (Same Origin Policy) bu bölümde ele alınacaktır. Ayrıca personal HTTP proxy'lerinin kurulum ve kullanımlarının yanı sıra bazı etkili Firefox güvenlik eklentilerine de değinilecektir.

2. Lab Ortamı

J2EE ve .NET çatılarının odak alındığı ve web uygulamaları ağırlıklı eğitimde labaratuar çalışmaları Eclipse ve MS Visual Studio geliştirme ortamlarında gerçekleştirilecektir. Örnek zafiyetleri barındıran özel yazılmış uygulamalar üzerinde farklı zafiyetler tespit edilecek ve bu zafiyetler dersler boyunca düzeltilmeye çalışılacaktır.

Bu bölümde lab ortamının kurulumları sağlanacak ve katılımcılara gerekli kaynaklar sağlanacaktır.

3. Güvenli Girdi Kontrolü

Güvenli girdi kontrolü yazılım güvenliğinde dikkate alınması gereken en önemli konudur. Bu bölümde yetersiz girdi kontrolü nedeniyle oluşabilecek zafiyetler örneklerle anlatılacak ve bu zafiyetlerin nasıl giderilebilecekleri ve oluşmadan önlenebilecekleri gösterilecektir.

3.1 Cross Site Scripting

XSS özellikle web uygulamalarında bulunan en popüler güvenlik zafiyetidir. Javascript ve diğer istemci taraflı dillerin saldırılarda kullanılması ile daha çok istemci tarafı hedef alınarak gerçekleştirilirler. Sistemlerin komple ele geçirilmesi, botnet oluşturma gibi ciddi sonuçlar oluşturabilen XSS, yazılım içerisinde düzeltilmesi en kolay anlaşılır açıklıklardan biridir.

Bu bölümde zafiyetin ciddiyeti laboratuvar ortamında örnekler ile anlaşıldıktan sonra, giderilmesi için kullanılması gereken stratejiler, API'lar anlatılacaktır. Özelde, MS AntiXSS ve OWASP-ESAPI-Java çatılarının XSS zafiyetini önlemek için nasıl kullanılabilecekleri demolar ile gösterilecektir.

Ayrıca, açıklık giderme 3rd parti API'lar olmaksızın, kullanılan çatıların (.NET MVC, .NET Klasik Web Sites, JSTL, Struts, Springs) XSS önlem tekniklerine değinilecektir.

3.2 SQL İnjection

SQL İnjection, günümüzde web uygulamaları yolu ile veritabanında saklanan verilerin çalınması riskinin arkasında yatan en önemli zafiyettir. Özellikle web uygulama dillerinin olgunlaşmadığı ilk yıllarda (1990'ların sonu ve 2000'lerin başı) güvenlik algısı son derece düşük geliştiriciler tarafından yetersiz girdi denetimi yapılması ve dinamik sql sorgular ile bolca sql injection zafiyeti oluşmuştur.

Bu bölümde sql injection zafiyeti kullanılarak neler yapılabileceği anlaşıldıktan sonra, giderilmesi için kullanılması gereken stratejiler, API'lar anlatılacaktır. J2EE uygulamaları için OWASP-ESAPI-Java DB metotlarından bahsedildikten sonra "sql query escape" çözümlerinin neden olabileceği problemlere değinilecektir.

SQL enjeksiyonunu kökten engellemek adına kullanılabilecek prepared statements, parameterized stored procedures ve bazı ORM çözümlerinden (Linq2Sql, Hibernate, v.b.) de bahsedilecektir.

3.3 Diğer Yetersiz Girdi Kontrolü Zafiyetleri

Bu bölümde özellikle web uygulamalarında oluşabilecek aşağıdaki zafiyetler hakkında bilgiler verildikten sonra, önleyici stratejiler anlatılacak ve farklı çatılar için farklı kontrol örnekleri verilecektir.

- Güvensiz File Upload

  İstemci tarafından sunucu tarafına dosya yüklemeler, yüklenilen dosya üzerinde muhtemel yetersiz kontroller nedeniyle ciddi güvenlik problemlerine yol açarlar.

- Code Injection

  Kod enjeksiyonu, kod bütününün veya parçalarının uygulamaya başarılı bir şekilde gönderilmesi ve daha sonra uygulama tarafından sanki kendi proje parçasıymış gibi çalıştırılmasıdır. En popüler kod enjeksiyonu PHP teknolojisinde bulunan ve bir çok sunucunun saldırganlar tarafından ele geçirilmesini sağlayan RFI (Remote File Inclusion) ve LFI (Local File Inclusion) zafiyetleridir.

- İşletim Sistemi Komut Enjeksiyonu (OS Commanding)

  İşletim sistemi komut saldırısı, uygulama kullanılarak yetkisiz bir şekilde rasgele işletim sistemi komutlarının çalıştırılması ile ilgili bir saldırı vektörüdür ve uygulamalar geliştirilirken yeterli veri doğrulama yapılmaksızın harici programları çalıştırma ihtiyacından doğar.

- CR/LF Injection

  Güvensiz kaynaklardan alınan veriler HTTP başlıkları oluşturmak için kullanılırlarsa, bu veriler içinde bulunabilecek yeni satır karakterleri (CR/LF), HTTP cevabının yapısını değiştirebilir ve örneğin kullanıcıya uygulamadan geliyormuş gibi cookie'ler gönderebilir.

- LDAP Injection

LDAP işlemleri için güvenilmeyen kaynaklardan gelen veriler denetimden geçirilmeden kullanılırsa, LDAP sorgularının yapıları değiştirilebilir ve yapıları değiştirilen LDAP sorguları LDAP sunucu üzerinde çalıştırılabilir.

- Xpath Injection

XML üzerinde sorgu özelliğini gerçekleştiren Xpath gibi sorgulama standartları yardımı ile oluşturulan sorgular, güvenilmeyen kaynaklardan gelen girdiler ile oluşturulursa, istenilen sorgular XML üzerinde çalıştırılıp cevapları alınabilir.

Bu bölümde yukarıda anlatılan muhtemel güvenlik zafiyetleri için özel geliştirilebilecek denetim stratejileri ve kullanılan çatıların sağladıkları API'lar anlatılacaktır.

3.4 Girdi Kontrolü Stratejileri

Terimler

Girdi Kontrolü stratejilerine geçmeden önce, denetim esnasında kullanılan terimlerin açıklanması gerekmektedir.

Beyaz Liste Girdi Denetimi (Whitelist):

Pozitif girdi denetimi olarak da geçer. Yazılım gereksinimleri ve dizaynı tarafından belirlenen veri tipleri için bilinen ***iyi*** karakterlerin veya karakter dizgilerinin kabul edilmesidir. Kabul edilen dizgiler belirli terimler ile ifade edilebileceği gibi regular expression'lar ile de ifade edilebilirler. Örnek olarak, kullanıcının kendi kredi kartları numaralarını uygulamamıza göndermesi halinde, gönderilen kart numarası kullanıcının kredi kartları kümesine karşı kontrol edilebileceği gibi, kredi kartı regular expression'ınına karşı da kontrol edilebilir. İlk kontrol hem denetim hem de yetkilendirme açısından daha güvenlidir.

Kara Liste Girdi Denetimi (Blacklist):

Negatif girdi denetimi olarak da geçer. Bilinen ***kötü*** karakterlerin veya karakter dizgilerinin reddedilmesidir. Genellikle beyaz liste çözümü yetersiz kalırsa uygulanabilir. Ancak gelişen ve yeni çıkabilecek atlatma teknikleri bu yöntemin en büyük düşmanlarıdır. Bugün yazılan bir karalistenin yarın atlatılamayacağının garantisi yoktur. Karaliste uygulamasına örnek olarak, kullanıcı adı için alınan verinin içinde virgüller hariç <,>,' karakterleri olması durumunda, verinin reddedilmesidir.

Temizleme İşlemi (Sanitize):

Arıtma işlemi olarak da geçer. Bir girdiyi kabul etmek veya reddetmek yerine, kabul edilebilir başka bir formata çevirilmesi işidir. İki şekilde gerçekleşebilir, beyazliste mantığı ile arıtma ve karaliste mantığı ile arıtma. Örneğin bir telefon numarası içinde geçen sayı olmayan bütün karakterlerin silinmesi beyazliste mantığı ile arıtmadır. Diğer taraftan bir girdi içerisindeki bütün tek tırnakların silinmesi karaliste mantığı ile yapılan bir arıtma olacaktır.

Kodlama İşlemi (Encode):

Girdilerin içindeki özel karakterlerin bir formattan başka bir formata değiştirilmesidir. Kodlama tersine çevrilebilir bir işlemdir. En bilinen kodlama çeşitleri HTML, URL kodlamalardır. Kodlama işleminde önemli olan şey, kodlama gerçeklendikten sonra hedef yorumlayıcı (interpreter) için özel karakterlerin önemini yitirmiş olmasıdır. Kodlama, yorumlayıcıları hedef alan bir çok saldırının (XSS, SQL Injection, OS Commanding, LDAP Injection v.b.) en etkili çözüm yoludur. İki defa yapılan kodlama veya kod çözme (decoding) atlatma yöntemleri için birebirdir ve kaçınılmalıdır.

Normalizasyon İşlemi (Canonicalization):

Normalizasyon (canonical) herhangi bir kodlamadan geçirilmiş bir dizginin en basit, en temel haline çevrilmesini anlatmaktadır. Farklı seviyelerde farklı anlamları olan normalizasyon, karmaşık bir işlemdir. Ancak normalizasyon kritik bir işlemdir çünkü saldırganlar denetim metotlarını atlatıp

yorumlayıcılarda çalışabilen kodlanmış girdiler üretebilmektedirler. İki defa kodlanmış bir girdi üretmek normal kullanıcıların sergileyebileceği bir davranış olmadığı gibi saldırı olarak algılanmalıdır. Uygulamalarda güvenilmeyen kaynaklardan gelen girdilerin denetlendiği metotlar normalizasyon işlemini uygulamış olmalıdırlar.

Normalizasyon işlemine en iyi örneklerden biri işletim sistemi dizin gösterimleridir;

```
/../../etc/passwd
/etc/passwd
```

Yukarıdaki iki yol da aynı dosyayı işaret etmektedir. Ama ikincisi passwd dosya yolunun en temel gösterim şeklidir ve dolayısıyla ilk dizinin normalize edilmiş halidir.

Kaçış İşlemi (Escape):

Kaçış işlemi aynı zamanda çıktı denetimi (output encoding) olarak da bilinir. Çoğu durumda kodlama ile eş anlamda kullanılır. Örnek olarak URL kodlama için hem encoding hem de escaping terimleri kullanılmaktadır.

Kaçış işlemine en güzel örnek, SQL Injection saldırılarından korunmanın bir yolu olan kullanıcı girdileri ile beraber dinamik olarak oluşturulan sql sorgu cümlelerinde, Oracle veritabanında tek tırnak işaretinin önüne bir tek tırnak daha gelecek şekilde değiştirilmesidir. Bu şekilde enjekte edilip sorgunun yapısının değiştirilmesinin önüne geçilebilmektedir. *(Kaçış işlemi her dinamik olarak oluşturulan sql sorguları için geçerli bir önlem yöntemi değildir.)*

Stratejiler

Standart Gösterim > Beyaz Liste > Kara Liste > Kodlama / Kaçış

Girdi denetimi yazılım içerisinde dört aşamadan oluşabilir; temele indirge, kısıtla, filtrele, temizle.

1. Standart Gösterim

İlk aşamada girdiler normalizasyon işleminden geçirilmelidir. Burada önemli olan, denetim sistemlerini atlatmak için kullanılan "iki defa / üç defa kodlama", "farklı format, aynı anlam" saldırılarına karşı girdiyi en temel haline getirmektedir.

2. Beyaz Liste

İkinci aşamada güvenilmeyen girdilerin uzunluk, tip ve karakter aralığı özelliklerine göre kısıtlanmasıdır. Bu kısıtlama alınan girdiye göre değişmeli ve beklenilenin dışındaki tüm verilerin reddedilmesi şeklinde geliştirilmelidir.

3. Kara Liste

Üçüncü aşama kendi başına tercih edilmese de, ikinci aşamadan sonra uygulanması uygun olabilir. Bu aşamada bilinen kötü amaçlı girdi listeleri, güvenilmeyen girdilerin içinde aranır ve yakalandığında bu girdiler reddedilirler. Atlatılması diğer aşamalara göre daha olası bir aşamadır.

4. Kodlama / Kaçış

En son aşama ise muhtemel zararlı girdinin zararsız hale getirilmesidir. Bu işlem, muhtemel zararlı karakterlerin (%00,%0d, %0a, v.b.) girdi içerisinden temizlenmesi veya girdinin gideceği kaynaklara

göre kodlama işleminden geçirilmesi (html, css, javascript, sql, ldap, os kodlamaları) şeklinde olabilir.

Girdi denetimi uygulamanın geneline yaygınlaştırıldığında kapsamlı bir güvenlik sağlarken, uygulamalarda birden fazla girdi denetimi stratejisi kullanmak mümkündür. Bu stratejilerin bazıları güvensizdir.

Girdi denetimi sırasında çoğu zaman regular expression'lar kullanılmaktadır. Ancak güvenlik için yazılan regular expression'ların farklı durumları kapsaması için gittikçe karmaşıklaştırılması güvenliği tersine yönden etkilerler. Çünkü karmaşık regular expression'ların kapsadığı (veya kapsamadığı) bütün kontrollerin listelenmesi kolay bir iş değildir.

Ayrıca girdi denetiminde karaliste mantığı (kötü görülenin reddedilmesi) ilk akla gelen stratejilerden biridir. Ancak kara listelerin tam kapsamlı olması son derece düşük bir ihtimal olduğundan güvenliği ters yönde etkileyecektir.

Yine girdi denetim kodlarının uygulama içerisinde farklı farklı yerlerde yazılması, algoritmalarının daha sonra kontrolleri açısından zor bir durum oluşturacaktır.

Bu bölümde, girdi denetimi gibi zor ve çetrefilli bir konu için kullanılabilecek güvenli stratejilerden bahsedilecektir. Ayrıca farklı çatıların sağladığı girdi denetimi API'ları listelenecek ve değerlendirilecektir.

4. Güvenli Kimlik Doğrulama

Kimlik doğrulama, yaptığı iş veya taşıdığı veri yönü ile hassas web uygulamalarında, kullanıcının gerçekten tanımlı kullanıcı olup olmadığı kontrolünde önemli bir rol oynar. Bu bölümde web'de kullanılan kimlik doğrulama tipleri ve uygulanması gereken önlemlerden bahsedilecektir.

4.1 Kimlik Doğrulama Çeşitleri

Genel olarak web uygulamalarında kullanılan kimlik doğrulama standartları, avantajları, dezavantajları, güvensiz kullanım durumları anlatılacaktır;

- Basic Authentication
- Windows Authentication
- OpenID
- Forms Authentication
- IP Kısıtlama

4.2 Sözlük/Kaba Kuvvet Kontrolü

Hedef kimlik doğrulama metotlarına uygulanabilecek en önemli kontrollerden iki tanesi sözlük ve kaba kuvvet denemeleridir. Bu denemelerin amacı ardarda denenmek üzere veriler gönderilmesi ve gelen cevabın incelenmesi yolu ile doğru verinin bulunmasıdır. Doğruluğu sınanmak istenen en önemli ve yaygın veriler kullanıcı adı ve şifrelerdir.

Yatay, dikey, diagonal ve üç boyutlu taramalar şeklinde türlere ayrılabilecek bu denemeler anlatılacak ve kullanılabilecek bazı araçların varlığı gösterilecektir.

Güvenli CAPTCHA Kullanımı

CAPTCHA, insanları bilgisayarlardan ayıran açık tam otomatik Turing testidir. Web uygulamalarında kullanılma amacı, bir bot tarafından otomatik ardarda isteklerin gönderilebilmesinin engellenmesidir. Uygulama, gönderilmesi beklenen istek ile beraber kullanıcının bir resim içerisindeki karakterleri de göndermesini bekler.

CAPTCHA uygulamalarındaki zafiyetler iki sınıfa ayrılabilir;

1. Kırılması kolay CAPTCHA resimleri
2. Zayıf CAPTCHA akışları
    a. CAPTCHA Replay saldırısı
    b. Zayıf CAPTCHA mantıkları (kısıtlı soru/cevaplar, basit metamatiksel denklemler, v.b.)
    c. CAPTCHA kütüphaneleri güvensiz kurulumu
    d. ...

Bu bölümde güvenli CAPTCHA uygulama stratejileri listelenecektir (örn: kırılması kolay CAPTCHA resimleri ile ilgili teknikler gösterilecek ve kırılması zor resimler üretilmesi için kullanılabilecek teknikler anlaşılacaktır).

Kaba Kuvvet Önleme Algoritmaları

Web uygulamalarına ard arda gönderilebilecek isteklerin saldırı olup olmadığının anlaşılması ve saldırının tespiti sonrası atılacak adımların belirlenmesi oldukça karmaşık bir iştir. Bazı durumlarda bu algoritmalar istenmeyerek hizmet dışı bırakma saldırılarına yol açabilir. Otomatik saldırılarda kullanılabilecek örnek temel kriterlerden ve bu saldırıları algılayabilecek temel algoritmalardan bahsedilecektir.

4.3 Kimlik Doğrulama Stratejileri

Kimlik doğrulama metotlarında uygulanan stratejiler güvenliği doğrudan etkilemektedir. Stratejiler arasında;

- Pozitif ve negatif kimlik doğrulamalar
- Kimlik doğrulama sonrası kullanılan HTTP cevap kodları
- Güvenli şifre reset algoritmaları

5. Güvenli Oturum Yönetimi

Web uygulamalarında kullanılan HTTP protokolü durum bilgisini tutmayan bir protokoldür. Diğer bir deyişle, aynı kullanıcıdan çıkan iki istek arasında web uygulaması için hiç bir ilinti yoktur. Fakat web uygulamalarının yararlı olabilmesi için durum bilgisi, yani her kullanıcının HTTP istekleri boyunca tanınması için gereken bilgi, dolayısıyla oturum yönetimi implementasyonlarının denetimi hayati önem taşır. Oturum yönetimi altyapısı genellikle kullanılan çatılar tarafından sağlanır, bu altyapıları kullanmak daha güvenilir bir uygulama oturum yönetimine sebep olacaktır.

5.1 Önceden Belirlenmiş Oturum (Session Fixation)

Uygulamaların oturum yönetimi sistemlerine yönelik saldırılardan bir tanesi de önceden belirlenmiş oturum (Session Fixation) saldırısıdır. Bu bölümde genellikle Java uygulamalarında ve J2EE containerlarında bulunan bu zafiyet ve çözüm önerileri anlatılacaktır.

5.2 Cross Site Request Forgery (CSRF)

Uygulamaların oturum yönetimi sistemlerine yönelik saldırılardan en önemli bir tanesi de oturum sürme (Cross Site Request Forgery) saldırısıdır. Bu bölümde CSRF zafiyeti ve çözüm önerileri anlatılacaktır. Ayrıca OWASP Java CSRFGuard ve ASP.NET ViewStateUserKey gibi önlem çözümlerinden bahsedilecektir.

5.3 Oturum Uygulama Zayıflıkları (Session Puzzling/Overriding)

Oturum mekanizması, sunucu tarafında bilgilerinin saklanarak, kullanıcıların güvenli bir şekilde tanınmasına ve işlemlerinin güvenli bir şekilde gerçekleştirilmesine olanak sağlar. Oturum değişkeni,

kullanıcı uygulamayı kullandıkça (tarayıcısını kapatmadıkça, çıkış işlemi gerçekleşmedikçe, v.b.) süreçte kalır. Bu süreç içerisinde uygulama oturumda kullanıcıya özel değişkenler oluşturur, bu değişkenleri yeniler veya siler. Bu değişkenlerin kontrolsüz olarak aynı oturum sürecinde farklı zamanlarda farklı amaçlar ile kullanılması, kritik zafiyetlere (kimlik doğrulama mekanizmasının atlatılması, izlenmesi zorunlu tutulan akışların atlatılması, hak yükseltme, içerik hırsızlığı, v.b.) yol açabilir. Bu bölümde bu zafiyetlerden bazıları ayrıntılandırılacak ve oturum uygulama hata pattern'leri incelenecektir.

5.4 Oturum Korsanlığı

Oturum korsanlığı genellikle trafik dinleme, araya girme ve XSS saldırıları nedeniyle uygulama kullanıcılarının cookie bilgilerinin çalınması sonucu oluşur. Özellikle XSS saldırıları nedeniyle çalınabilecek cookie bilgilerinin nasıl korunabileceği ile ilgili metotlar anlatılacaktır.

5.5 Oturum Yönetimi Stratejileri

Bu bölümde uygulamaların oturum yönetimi esnasında kullanılabilecekleri stratejiler anlatılacaktır. Bu stratejiler arasında güvenli bağlantı sağlanması, cookie değerlerinin kriptoğrafik olarak korunması, oturum geçerlilik süreleri konuları vardır.

6. Güvenli Yetkilendirme

Kullanıcıların kimlikleri doğrulandıktan sonra uygulamanın belli özelliklerini kullanmak veya belli kısımlarına erişmeleri gerekecektir. Kaynaklara erişimlerin kısıtlanması, yetkilendirme kontrollerinin uygulanması ile gerçekleştirilir.

6.1 Directory Traversal

Birçok web uygulaması ve web/uygulama sunucusu görevlerinin bir kısmında dosya işlemleri ile uğraşmak zorunda kalırlar. Eğer bu işlemlerde özellikle dosya ismi ve yolu verilerinde yetersiz kontroller yapılırsa, saldırgan sunucunun/uygulamanın çalıştığı dosya sistemi içinde gezinme yetkisini kazanabilir. Uygulamadan dizin ve dosyalara güvenli erişim, ve güvensiz kaynaklardan alınan veriler üzerine uygulanabilecek canonicalization (en temel ve basit hale indirgeme) teknikleri bu bölümde anlatılacaktır.

6.2 Güvensiz Direk Nesne Referansı

Uygulamalar kullanıcılardan gelen istekleri genellikle veritabanı, dosya sistemi gibi kaynaklardan karşılarlar. Bu eşleşme dolaylı değil de direk olduğunda, aynı zamanda bütünlük ve yetkilendirme kontrolleri yetersiz kaldığında kullanıcıların diğer kullanıcıların bilgilerine erişmesine yol açılır. Eşleşme esnasında uygulamalar, erişilecek bilgiler üzerinde kontroller gerçekleştirmelidirler. Bu bölümde sistem kaynaklarına güvenli direk ve dolaylı referans teknikleri anlatılmaktadır.

6.3 Açık Yönlendirmeler (Open Redirects)

Açık yönlendirmeler bir web uygulamasının kullanıcıdan aldığı harici URL bilgisini bir HTTP / HTML / Javascript yönlendirme (redirect) tekniğinde kullanmasıdır. Bu zafiyet kullanıcıların uygulamaya olan güveni yolu ile kandırılmasına yol açabilir. Açık yönlendirmenin önüne geçmek için kullanılabilecek kısıtlamalar ve riskler üzerinde durulacaktır.

6.4 Forceful Browsing

Web üzerinden kaynaklara genellikle URL ile erişilir. Bazı kaynaklar üzerindeki yetkilendirme kontrolünün gizlilik esasına dayandırılması yanlışına düşülmesi mümkündür. Gizlilik ile korunan URL'ler tahmin veya açığa çıkmış kayıt dosyaları yardımı ile saldırganlar tarafından bulunabilir. Bu durumda erişime kısıtlandırılmaya çalışılmış kaynaklara ulaşım çok kolay olacaktır. Bu bölümde yetkilendirme kontrolünün kapsamlı olması için gerekli adımlar anlatılacaktır.

6.5 Yetkilendirme Stratejileri

Bu bölümde, uygulamalarda rastlanan hatalı rol tabanlı yetkilendirme kontrolleri stratejilerinden bahsedilecektir. Ayrıca kapsamlı bir yetkilendirme için gerekli stratejiler de bu bölümde listelenecektir. Örnekler;

- Sıklıkla rastlanan hatalı rol tabanlı yetkilendirme kontrolü uygulamalarından en önemlisi rol bilgilerinin yazılım içinde direk kullanılmasıdır. İzlenmesi gereken strateji, erişim kontrolleri sırasında var olan bilgilerin (kaynak ve kaynak üzerinde uygulanacak operasyon) kullanılarak, denetimin gerçekleştirilebilmesidir.
- Minimum hak prensibine göre yetkilerin kullanıcılara gereğinden fazla verilmemesi ve uygulama çatısının kaynaklara erişim kısıtlama yöntemlerinin mümkün olduğunca uygulanmalıdır. (Java SecurityManager ve ASP.NET CAS, Code Access Security).
- Klasik HTTP isteklerinde yanında AJAX, Flash kaynaklı isteklerinde kontrolden geçirilmelidirler.

7. Güvenli Dizayn

Uygulama geliştirme süreçlerinde dizayn çok büyük bir önem taşır. Bir çok güvenlik problemi kod yazım sürecine geçilmeden bu safhada fark edilebilirse çok küçük eforlarla düzeltilebilir. Ayrıca bu safhada kontroller, yardımcı kütüphaneler, uyulması gereken standartlar, v.b. güvenlik önlemleri konuşularak stratejik bir şekilde uygulanmalıdır.

Genel bir kural olarak, bir yazılım hatasının gereksinimlerin belirlendiği safhada düzeltilmesi 1TL iken, bu hatanın dizayn safhasında düzeltilmesi 10TL, yazılım safhasında düzeltilmesi 100TL ve üretim aşamasında (production) düzeltilmesi 1000TL'dir. Bu nedenle yazılım problemlerinin en erken safhalarda bulunması ve düzeltilmesi finansal ve iş gücü anlamında projelere büyük katkıda bulunacaktır.

7.1 Güvensiz İş Mantığı Kontrolleri

Yazılımlarda bulunan güvenlik hataları iki genel kategoriye ayrılabilir; kod hataları ve dizayn hataları. Kod hataları geliştirme safhasında yapılan genel olarak syntax problemleridir. Dizayn problemleri ise geliştirme ve daha önceki safhalarında ortaya çıkan problemlerdir.

Dizayn problemlerinin diğer bir adı iş mantığı zafiyetleri olarak geçer. İş mantığı güvenlik zafiyetleri, yetersiz geliştirilmiş veya yazılmış algoritmalardan ve dizayn elementlerinden kaynaklanır.

7.2 Tehdit Modelleme

Uygulamalarda var olabilecek güvenlik zafiyetlerinin bulunması için gerçekleştirilen bir teknik de, tehdit modelleme olarak adlandırılır. Tehdit modelleme aslında geliştirmenin her safhasında yapılabilir ama problemlerin erken safhalarda fark edilmesi için geliştirme başlamadan önceki her safha uygun olacaktır.

Tehdit modelleme farklı proje sorumlularının bir araya gelerek yaptıkları toplantılar sonrası uygulamanın barındırdığı veya barındıracağı tehditlerin bulunması, listelenmesi ve önleyici adımların atılmasıdır. Tehdit modelleme;

- uygulamanın daha iyi anlaşılmasını
- bulunmuş ve detaylanmış tehditlerin QA yetkililerinin elinde test edilebilmesini
- kurumun tehdit haritasının çıkarılmasını
- güvenlik problemlerinin erken safhalarda düzeltilerek finansal ve iş gücüne olumlu etki etmesini

sağlar.

Bu bölümde tehdit analizinin temel elementleri, kullanılabilecek yazılım araçları ve örnekler anlatılacaktır.

8. Güvenli Hata Yönetimi ve Kayıt Tutma

Uygulamalar beklenmedik durumlarla karşılaştıklarında hata üretirler. Üretilen hatalar, kaynak problemi bulabilmek için detaylı olmak zorundadırlar. Ancak bu detaylı hata mesajlarını bütün kullanıcılar görmemelidirler. Bu durum, bilişim güvenliğinde önemli yer tutan "bilmesi gereken" prensibi olarak bilinir.

Diğer yandan uygulamalar önemli aksiyonları daha sonra işlenebilmesi ve anlamlandırabilmeleri için kayıt altına almak zorundadırlar. Bu kayıtlar içinde kullanıcıdan alınan girdiler olacağından, bu girdilerin denetimden geçirilerek kayıt altına alınması uygun olacaktır.

8.1 Yetersiz Hata Yönetimi

Sistem ve uygulama üzerinde yönetici yetkisi sahibi olmayan kullanıcılara uygulamanın detaylı hata mesajı dönmesi "bilmesi gereken" prensibine aykırıdır. Bu detaylı bilgiler daha ileri seviye saldırı düzenlenmesinde kullanılabilir. Bu nedenle uygulama detaylı hata mesajlarını sadece yetkili kullanıcılara (geliştirici, qa, sistem yöneticisi v.b.) göstermeli ve diğer bütün kullanıcılara detaysız hata mesajları dönülmelidir.

8.2 Try/Catch/Finally Blokları

Yazılımda hata ayıklamak/yakalamak için kullanılan yapılar try/catch bloklarıdır. Bu yapılar yanlış kullanıldığında kontrolsüz kaynak tüketimine, detaylı sistem hata mesajlarının kullanıcılara ulaşmasına sebep olabilir. Bu bölümde try/catch/finally bloklarının örnek kullanımlarına yer verilecektir.

8.3 Kayıt Sahteciliği

Güvenilmeyen kaynaklardan alınan girdiler uygulama tarafından oluşturulan kayıtlar içinde yer bulabilirler. Örnek olarak dosya sistemine yazılan her tek satır kayıt, saldırganın girebileceği ve denetlenmeyen "yeni satır" (%0d%0a) karakterleri ile extra bir satıra kayabilir, bu durum da kayıtların bütünlüğünü bozabilir. Kayıt altına giden güvenilmeyen kaynaklardan gelen bilgiler denetimden geçirildikten sonra kayıt kaynağına aktarılmalıdır.

8.4 Güvenli Hata Yönetimi ve Kayıt Tutma Stratejileri

Hata yönetiminde "bilmesi gereken" prensibi mutlaka uygulanmalıdır. Bunun yanında detaylı hata mesajları yerinde genel kullanıcı hata mesajları üretilmelidir. Ayrıca yazılım içinde hata yakalama kod blokları bazı saldırıları önlemek amacı ile en iyi yöntemlere uygun yazılmalıdır.

Şifreler, kredi kart numaraları, adresler, telefon numaraları gibi kullanıcı mahremiyetini etkileyen hassas bilgiler kayıt altına alınırken bir maskelemeden geçirilmeli ve kesinlikle oldukları gibi kayıt kaynaklarına aktarılmamalıdır.

Kayıt altına alınacak bilgiler daha sonra doğru bilgilendirmeyi sağlacak önemli verileri barındırmalıdır (İstemcinin gerçek IP'si, zaman, sınıf, metot, kayıt seviyesi, kullanıcı adı, yapılmak istenen aktivite ismi, v.b.).

Yönetim aktiviteleri (yapılandırma ayar değişiklikleri, kullanıcı ekleme/çıkarma/değiştirme v.b.) ve önemli uygulama aktiviteler (yetkilendirme, kimlik doğrulama, iş mantığı işlemleri v.b.) başarılı veya başarısız mutlaka kayıt altına alınmalıdır. Ve kayıt işlemleri merkezi bir API yardımı ile gerçekleştirilmelidir.

9. Güvenli Kriptografi

Uygulamalar veri bütünlüğü ve gizliliği gibi gereksinimleri kriptoğrafik API'lar kullanarak karşılarlar. Bu gereksinimler;

- Bütünlük koruması – *Hash ve Salted Hash*
- Rasgele Sayı ve Dizgi Üretme - Randomizer
- Simetrik ve Asimetrik Şifreleme – *Symmetric & Asymmetric Encryption*

Örnek vermek gerekirse;

- Bütünlük koruması ile şifre saklamak ve geçerliliği denetlemek güvenli bir şekilde gerçekleştirilebilir.
- Rasgele karakter dizgiler ile oturum ID'leri, dosya isimleri, bir kerelik kullanım için şifreleri güvenli bir şekilde üretmek mümkün olabilir.
- Simetrik şifreleme yardımı ile verilerin saklanırken ve/veya transfer esnasında gizlilikleri ve bütünlükleri güvenli bir şekilde sağlanabilir.

9.1 Kriptoğrafi Giriş ve Güvenli Algoritmalar

Bu bölümde bütünlük, şifreleme ve rasgele sayı üreten güvenlik algoritmalarına kısa bir giriş yapılacak ve FIPS 140-2 standartı tarafından sıralanan güvenli algoritmalar listelenecektir.

9.2 OWASP ESAPI Java - Encryptor / Randomizer

Java güvenli kriptoğrafi işlemleri için OWASP ESAPI-Java'nın sağladığı API'lar gözden geçirilecektir. API'ın şifreleme işlemleri öncesi kullanılması için yapılması gereken yapılandırma ayarlarından bahsedilecek ve şifremele metotları örneklendirilecektir.

9.3 .NET Çatısı - Security.Cryptography

.NET güvenli kriptoğrafi işlemleri için .NET çatısının sağladığı API'lar gözden geçirilecektir. Şifreleme metotları örneklendirilecektir. Ayrıca süreci işleten kullanıcın veya sürecin üzerinde koştuğu bilgisayar hesabının bilgilerinin otomatik olarak kullanılarak Data Protection API (DPAPI) ile verilerin nasıl şifrelendiği ve çözüldüğü anlatılacaktır.

DPAPI'ın, şifreleme işlemlerinin en kritik noktalarından olan anahtarın güvenli bir şekilde üretilmesi ve saklanması konularında ne gibi kolaylıklar sağladığı gösterilecektir.

10. Statik Kaynak Kod Analizi

Geliştirme esnasında veya bittikten sonra barındırdığı güvenlik zafiyetlerini çıkarmanın en etkili yollarından biri statik kod analizidir. Yazılmış kodlar üzerine yapılan bu denetim hem manuel hem de otomatik olarak gerçekleştirilebilir. Ancak kod tabanı ve genişliği düşünüldüğünde manuel analizler hızlı olmayacaktır. Yine de otomatik testler sonrası, akıllı manuel denetimler false pozitifleri eleme ve iş mantığı problemlerini bulma konularında çok etkilidir.

10.1 Statik Kaynak Kod Analizi Temelleri

Statik kod analizi, ürettikleri programların çalıştırılmadan yapılan analizlerdir. Bu analizler, bazı durumlarda direk kaynak kod üzerinde çalıştırılabildiği gibi, bazı durumlarda da kaynak koddan üretilen ara kodlar (intermediate languages) üzerine çalıştırılabilirler.

Amaç, kaynak kod içindeki hataların bulunmasıdır. Hataların bulunması en basit olarak tehlikeli olarak görülen metotların regular expression yardımı ile kod içinde aranmasından, kaynak kodun özel derleyiciler sayesinde daha analiz edilebilir hale getirilerek farklı yöntemler uygulanmasına kadar değişebilir.

Kaynak kod analizinde görülebilecek yaklaşımlar; veri akışları, kontrol akışları, yapılandırma problemleri, yapısal problemler, v.b. olarak belirtilebilir.

10.2 Statik Kaynak Kod Analiz Araçları

Bu bölümde bazı ticari olmayan statik kaynak kod analiz araçları incelenecektir. Bu araçlar kaynak kod üzerinde koşulduklarında, kurallarını barındırdıkları güvenlik zafiyetlerini kaynak kod içinde bulup raporlarlar. Analiz edilecek kaynak kod araçları;

- CAT.NET: .NET projelerinde kullanılabilecek Microsoft temelli bir güvenlik statik kaynak kod aracıdır.
- LAPSE+: Java projelerinde kullanılabilecek Stanford Üniversitesi doktora tezi temelli bir güvenlik statik kaynak kod aracıdır.

11. Güvenli Web Servisleri ve AJAX

11.1 Web Servisleri Zafiyetleri ve Önlemleri

Web Servisleri ve SOA (Service Oriented Architecture) uygulamaları birer servis gibi davranarak işlerliklerini arttırmayı amaçlamaktadırlar. XML teknolojisinin ağırlıklı olarak kullanıldığı web servisleri, temel olarak HTTP/HTTPS protokollerini kullanırlar. İş çevrelerinin Web'e açılım süreçlerinde dahili servislerini bu teknoloji ile dışarı açabilmeleri yetersiz kontroller ile beraber büyük bir tehlike oluşturmaktadır.

En önemli web servis zafiyetleri yetersiz yetkilendirme, kimlik doğrulama konularıdır. Ayrıca XML tabanlı zafiyetlerin de, üretilen kod içerisinde yönetilmesi her zaman mümkün olmasa da bilinmesi bu tür uygulamaların riskleri hakkında bilgilendirici olmaktadır.

11.2 Güvenli Web Servisleri Stratejileri

Uygulamalar için kullanılacak XML parser'ların (özel kütüphaneler veya kullanılan SDK'lar) en son yamalarının uygulanmış olması sağlanmalıdır. Özellikle XML tabanlı hizmet dışı bırakma saldırıları (örn: XML Entity Expansion Attack v.b.) eski XML parser'ları hedef almaktadır.

Web servisi ile açılan bütün operasyonlar yetkilendirme ve kimlik doğrulama kontrollerinden geçirilmelidirler. Ayrıca web servisi ile açılan operasyonlardan gereksiz olanlar mutlaka kapatılmalıdır. Bu durum genellikle önceden yazılmış sınıfların metotlarını kodsal bir değişiklik yapılmadan web servisi olarak açabilen çatıları kullanan uygulamalar için geçerlidir.

Klasik web servislerinin en büyük güvenlik problemlerinden biri de kimlik doğrulama ve yetkilendirme yetersizlikleridir. Bu nedenle dışarıya açılan bütün hassas web servis operasyonlarını kimlikleri doğrulanmış kullanıcılar çağırabilmelidir. Ayrıca rolüne göre bir web servisi operasyonunu çağırabilen bir kullanıcı, diğer bir web servis operasyonunu çağıramayabilir.

11.3 AJAX Zafiyetleri ve Önlemleri

Son yılların en popüler standartlar bütünü Ajax (Asynchronous Javascript and XML) aynı zamanda Web 2.0 döneminin başlamasında etkili bir rol üstlenmiştir. Karmaşık yapısı ve geliştiricilerde yaratabileceği yanlış güvenlik hissi nedeniyle Ajax teknolojileri güvenlik zafiyetlerine yol açabilmektedirler.

En önemli AJAX zafiyetleri yetersiz yetkilendirme ve javascript tabanlı problemlerdir. Ayrıca kullanılan AJAX çatılarının güvenlik zafiyetleri olup olmadığı, varsa bu zafiyetlerin nasıl adreslenmesi gerektiği önemli noktalardır.

11.4 Güvenli AJAX Stratejileri

Kullanılan AJAX kütüphanelerinin güvenlik yamaları çıkmış ise mutlaka yenilenmelidirler. Bu tür çatılarda çıkan güvenlik problemleri ciddi saldırılara yol açabilmektedir.

AJAX isteklerinin klasik web istemci ve web sunucu arasındaki isteklerden farklı olmadıklarından yola çıkılarak yetkilendirme, kimlik doğrulama ve oturum yönetimi (CSRF önleme gibi teknikler) gibi güvenlik önlemlerinin alınması gerekmektedir.

Yeni istemcilerde düzeltilmiş olsa da JSON tabanlı veri alışverişinde güvenli formatlar kullanılmalıdır. Ayrıca güvenli Javascript yapıları kullanılmalıdır.

Bazı AJAX geliştirme ortamlarında (Google Web Toolkit gibi) üretilen kodların geliştirici açısından istemci ve sunucu farklılığı ortadan kalktığından güvenlik kontrol kodlarının nerede çalıştığı hissi kaybolabilir. Bu da girdi denetimi, yetkilendirme gibi güvenlik kontrollerinin tehlikeli bir şekilde istemci tarafında yapılmasına yol açabilir. Bu nedenle geliştiricilerin yazdıkları kodların hangi kapsamda ve ortamda çalışacakları bilgisini unutmamaları ve buna göre kod yazmaları gerekmektedir.