Girdi Denetimi

Amaç

İster kullanıcıdan, ister kullanılan altyapıdan, ister dış kaynaklardan veya veritabanlarından gelsin, uygulamanın her türlü girdi türüne karşı sağlam (bağışık) durmasını sağlamak.

Etkilenen Platformlar

Hepsi.

İlgili COBIT Konuları

DS11 – Veriyi yönet. Bütün bölümler gözden geçirilmelidir.

Açıklama

En yaygın web uygulama güvenlik zaafiyeti, istemciden veya çevresel kaynaklardan gelen girdinin yeterince denetlenmemesidir. Bu zaafiyet yorumlayıcı enjeksiyonu, locale/Uncode saldırıları, dosya sistemi saldırıları ve arabellek taşmaları gibi nerdeyse uygulamalardaki en ciddi açıklıklara yol açar.

İstemci veriyi değiştirmek için her türlü şansa sahip olduğundan istemciden veriye asla güvenilmemelidir.

Tanımlar

Bu dokümanda şu tanımlar kullanılmaktadır:

- **Bütünlük Kontrolleri**

  Verinin original hali ile aynı olduğunu ve değiştirilmediğini sağlamak

- **Denetleme**

  Verinin güçlü kontrollere tabi olduğunu, yazım kontrollerinin doğru olduğunu, uzunluğunun sınırlar içerisinde olduğunu, sadece izin verilen karakterler içerdiğini veya eğer sayısal ile doğru işarete sahip olduğunu ve tanımlı aralıklar içerisinde olduğunu sağlamak

- **İş kuralları (mantıkları)**

  Verinin sadece denetlenmeyip aynı zamanda iş mantığına gore doğru olduğunu sağlamak. Mesela, faiz oranlarının izin verilen seviyelerde olması.

Bazı dokümanlar ve referanslar bazı kavramları değişik kapsamlarda değişik anlamlarda kullanırlar ki ilgili herkes için bu durum çok karmaşıktır. Bu karmaşa organizasyonun devamlı finansal kaybına neden olur.

Bütünlük kontrollerini nerde uygulamalı

Verinin uygulamadan gizli bir alan (hidden field) içinde kullanıcının tarayıcısına veya dahili olarak kullanılan bir işlem IDsi olarak üçüncü parti bir ödeme kapısına gönderilmesi gibi veri ne zaman güvenilen bir sınırdan daha az güvenilen bir sınıra geçerse bütünlük kontrolleri uygulanmalıdır.

Bütünlük kontrolünün türü (checksum, HMAC, şifreleme, dijital imza) güven çemberini geçen verinin risk ile direk ilgili olmalıdır.

Denetlemeyi nereye koymalı

Denetleme her katmanda uygulanmalıdır. Ancak, denetleme kodu çalıştıran kodun fonksiyonuna göre uygulanmalıdır. Mesela, web / sunum katmanı web ile alakalı sorunlar için denetlerken, kalıcı (persistence) katmanlar SQL/HQL enjeksiyonu gibi, dizin bulmalar LDAP enjeksiyonları gibi sorunları denetlemelidirler.

İş mantığı denetlemesini nereye koymalı

İş mantıkları dizayn safhasında bilinirler ve gerçeklemeyi (implementation) etkilerler. Ancak, kötü, iyi ve “en iyi” yaklaşımlar mevcuttur. Çoğunlukla en iyi yaklaşım kodlamada en basit olanıdır.

Örnek - Seneryo

- Arka uç sisteminden sağladığınız bir hesap listesini oluşturuyorsunuz:
- Kullanıcı bir hesap seçiyor, bir borçlu hesap seçiyor ve ileriye basıyor.

Yanlış Yol

Hesap seçme opsiyonu (select option) direk olarak okunur ve hesap oluşturulan listeden biri mi değil mi kontrol edilmeden bir mesaj içerisinde arka uça iletilir.

Bu neden kötüdür:

Bir saldırgan HTML'i istediği şekilde değiştirebilir:

- Girdi denetimi yetersizliği arka uçtan alınabilecek hata mesajlarını almaya ve performansı düşürmeye neden olabilir.
- Arka uç, ön ucun kolaylıkla elimine edebileceği zararlı veri ile baş edemeyebilir. Mesela arabellek taşmaları, XML enjeksiyonu veya benzerleri…

Kabul Edilebilir Yöntem

Hesap seçimi parametresi sadece kod tarafından okunur ve daha once oluşturulmuş listeyle karşılaştırılır.

```
if ( account.inList(session.getParameter('payeelstid') ) {
backend.performTransfer(session.getParameter('payeelstid'));
}
```

Bu parameter manipülasyonunu engeller ama çoğu işi tarayıcının yapmasını sağlar.

En İyi Yöntem

Orjinal kod hesap isimleri yerine sadece indeksler <option value="1" ... > üretir

```
int payeeLstId = session.getParameter('payeelstid');
accountFrom = account.getAcctNumberByIndex(payeeLstId);
```

Bu sadece HTML'in daha kolay parse edilmesini sağlamaz, aynı zamanda denetleme ve iş mantığı denetlemelerini çok basit hale indirger. Alan artık manipüle edilemez.

Sonuç

Katmanlı güvenlik sağlamak ve rasgele girdileri denetlemeyen arka uç sunucular gibi güven sınırlarında saldırı dizgilerini engellemek için, iş mantığı denetimi yapılmalıdır.

Bu bütün iş mantığı kurallarının uygulanması gerektiği anlamına gelmemektedir – arka uçtan alınabilecek hata mesajlarını almamak ve manipüle edilmiş verileri bu arka uçlara göndermemek anlamına gelmektedir.

Girdi Denetimi Stratejileri

Veri denetiminde dört strateji vardır ve aşağıdaki sıra ile uygulanmalıdırlar:

Sadece iyiyi kabul et.

Bir posta kodu bekliyorsanız, posta koduna göre denetleme yapın (tür, uzunluk ve syntax):

```
public String validateAUpostCode(String postcode) {
      return (Pattern.matches("^(((2|8|9)\d{2})|((02|08|09)\d{2})|([1-
9]\d{3}))$", postcode)) ? postcode : '';
}
```

Bilinen kötüleri refüze edin. Eğer %3f veya JavaScript veya benzeri karakterler beklemiyorsanız bunları içeren dizgileri refüze edin:

```
public String removeJavascript(String input) {
      Pattern p = Pattern.compile("javascript", CASE_INSENSITIVE);
      p.matcher(input);
      return (!p.matches()) ? input : '';
}
```

Kötü amaçlı yazılımları engellemek için 90'a yakın düzenli ifade (regex) kullanmak gerekebilir ve her düzenli ifade her alanda uygulanmalıdır. Açıkça bu yavaş ve güvenli olmayacaktır.

Düzeltmek

Girdiyi "güvenli" hale getirmek amacı ile karakterleri çevirin (HTML kodlama veya çift tırnakları silmek gibi) veya engelleyin:

```
public String quoteApostrophe(String input) {
      return str.replaceAll("[\']", "&rsquo;");
}
```

Bu pratikte pek uygulanamaz çünkü bu kural bir çok istisnai durum vardır.

Denetim yok

```
account.setAcctId(getParameter('formAcctNo'));

...

public setAcctId(String acctId) {
      cAcctId = acctId;
}
```

Bu kendiliğinden güvensiz ve kullanılması kesinlikle önerilmez. İş her girdi denetimsiz örneği ortadan kaldırmalı çünkü girdi denetim yetersizliği genellikle uygulamanın, sunucunun ve ağ güvenlik kontrollerinin atlatılmasına neden olur.

Sadece "şu anda bilinen kötü"nün reddi eğer girdi dizgi ise yetersiz kalır. Bu strateji anti-virüs yenileme kavramı ile örtüşür. Uygulama "kötü" düzenli ifadelerin günlük olarak yenilemesine izin vermiyorsa ve her yeni saldırıyı düzenli olarak araştıran birini işe almıyorsa bu yaklaşım çok geçmeden atlatılır.

Hemen hemen bütün alanlar belli bir gramere sahip olduklarından, basit pozitif bir ifadeyi kontrol etmek, bir çok negative ifadeyi kontrol etmekten daha basit, hızlı ve güzenlidir.

Veri:

- Her zaman kuvvetli tipte (strongly typed) olmalıdır
- Uzunluğu kontrol edilmiş ve alanlarının uzunluğu minimize edilmiş olmalıdır.
- Eğer sayısal ile aralığı control edilmelidir.
- İşaretli olması gerekmedikçe işaretsiz (unsigned) olmalıdır.
- Syntax veya gramer ilk kullanımdan önce kontrol edilmelidir.

Kodlama ilkeleri, istemciden veya güvensiz kaynaklardan gelen girdi üzerinde bir kirlilik izleme mekanizması kullanmalıdır:

```
taintPostcode = getParameter('postcode');
validation = new validation();
postcode = validation.isPostcode(taintPostcode);
```

Parametre manipülasyonunu engelle

Bir çok girdi kaynağı vardır:

- REMOTE_ADDR, PROXY_VIA veya benzerleri gibi HTTP başlıkları
- getenv() veya sunucu özellikleri gibi çevresel değişkenler
- Bütün GET, POST ve Cookie verileri

Bunlara değiştirilmesine pek ihtimal verilmeyen radio buttonlar, drop downlar dahildir – İstemci taraflı her HTML saldırgan tarafından değiştirilebilir.

Yapılandırma verileri (hatalar yapılmış olabilir)

Harici sistemler (XML girdisi, RMI, web servisleri gibi her türlü girdi mekanizmasından gelebilecek)

Bütün bu girdi kaynakları güvenilmeyen girdi sağlarlar. Güvenilmeyen veri kaynaklarından alınan veriler ilk kullanımdan once güzelce denetlenmelidirler.

Gizli alanlar

Gizli alanlar, sunucuda durum bilgisini kullanmayı önlemek için basit yollardır. Bir çok sayfadan oluşan "sihirbaz" tarzı uygulamalarda özellikle kullanılırlar. Ancak, kullanım tarzları uygulamanızın dahili çalışma biçimini ifşa eder ve veriyi manipülasyon ve tekrar oynma saldırılarına karşı açık bırakır. Genel olarak, gizli alanlar sadece sayfa sayılarını tutmak için kullanın.

Eğer gizli alan kullanmak zorunda iseniz, işte bazı kurallar:

- Gizli bilgiler, mesela şifreler, kesinlikle açıkta gitmemelidir.
- Gizli alanların bütünlük kontrolleri yapılmalı ve tercihen sabit olmayan ilklendirme dizgileri **(initialization vectors)** ile şifrelenmelidirler. (yani değişik zamanlarda değişik kullanıcılar değişik rasgele ilklendirme dizgilerine sahip olmalıdırlar)
- Şifrelenmiş gizli alanlar tekrar oynama saldırılarına karşı bağışık olmalıdırlar.
- Kullanıcıya gönderilen veri sunucu üzerinde daha önceden denetlenmiş olsa bile son sayfa gelir gelmez denetlenmelidir – bu tekrar oynama saldırılarının risklerini düşürülmesinde yardımcı olur.

Tercih edilen bütünlük kontrolü en azından SHA-256 kullanan HMAC veya tercihen dijital olarak imzalanmış veya PGP kullanılarak şifrelenmiş olmalıdır. IBMJCE SHA-256'I destekler ama PGP JCE desteği Legion of the Bouncy Castle (http://www.bouncycastle.org/) JCE sınıflarını gerektirecektir.

Bu veriyi oturum bilgisinde geçici olarak tutmak daha basittir. Oturum nesnesini kullanmak en güvenilir yoldur çünkü bilgi kullanıcıya gözükmez, çok daha az kod, nerdeyse hiç CPU, disk veya I/O operasyonu, daha az hafıza (özellikle bir çok sayfadan oluşmuş formlar için) gerektirir ve daha az ağ tüketimi sağlar

Veritabanı tarafından desteklenen oturum nesnesi durumunda, büyük oturum nesneleri varsayılan yönetici parçasının yönetimi için çok büyük olabilir. Bu durumda, strateji denetlenmiş veriyi veritabanında tutmak ama işlemi "bitmemiş" şeklinde imzalamaktır. Her sayfa, bütün işlem bitene kadar bitmemiş işlemi yenileyecektir. Bu, veritabanı yükünü, oturum büyüklüğünü ve kullanıcılar arasındaki aktiviteleri azaltırken uygulama manipülasyona bağışık kalacaktır.

Gizli alanları kapsayan kod, kaynak kodu analizlerinde reddedilmelidir.

ASP.NET Viewstate

ASP.NET form verisini istemciye "Viewstate" gizli alanında yollar. Güvenli gibi dursa da, bu "şifrelenmiş hal" aslında açık text gibidir ve tarafınızdan aksi yapılmadıkça hiçbir bütünlük koruma kontrolü yoktur (ASP.NET 1.1). ASP.NET 2.0'da ise manipülasyon koruması varsayılı olarak açıktır.

Aynı mekanizmaya benzeyen herhangi bir uygulama çatısı hata yapabilir – uygulama çatınızı veriyi istemciye yollamak ile alakalı destek bilgilerini okumalısınız. Tercihen, bu veriler istemciye gitmemelidirler.

Saldırıya açık olup olmadığınızı anlama (yolları)

machine.config dosyasını inceleyin:

- enableViewStateMac "true" değilse, viewstate yetkilendirme durumu içeriyorsa tehlikedesinizdir.
- viewStateEncryptionMode "always" değilse, viewstate kimlik bilgilerinizi içeriyorsa tehlikedesinizdir.
- Diğer bir çok müşteri ile bir sunucuyu paylaşıyorsanız, ASP.NET 1.1'de aynı makine anahtarını (machine key) paylaşıyorsunuz. ASP.NET 2.0 ise uygulama başına farklı bir viewstate anahtarı oluşturabilirsiniz.

Kendinizi koruma (yolları)

- Eğer uygulamanız viewstate'ten değiştirilmeden dönen veriye göre çalışıyorsa, viewstate bütünlük korumasını (integrity) açmalı ve aşağıdakileri uygulamayı düşünmelisiniz:
- Her veri uygulama için hassasmışçasına viewstate'i şifreleyin
- Eğer paylaşılan bir servis üzerinde iseniz acilen ASP.NET 2.0' geçin
- Çok hassas viewstate verisini oturum değişkeninde tutun

Selects, radio buttonlar, ve checkboxlar

Bu bileşenlerin değerlerinin değiştirilemeyeceği yaygın olarak inanılan bir görüştür. Bu yanlış. Aşağıdaki örnekte, hesap ele geçirmeye varabilecek şekilde gerçek hesap numaraları kullanılmıştır:

```
<html:radio value="<%=acct.getCardNumber(1).toString( )%>"
property="acctNo">
<bean:message key="msg.card.name"
arg0="<%=acct.getCardName(1).toString( )%>" />
<html:radio value="<%=acct.getCardNumber(1).toString( )%>"
property="acctNo">
<bean:message key="msg.card.name"
arg0="<%=acct.getCardName(2).toString( )%>" />
```

Bu aşağıdakini üretir (örnek olarak):

```
<input type="radio" name="acctNo" value="455712341234">Gold Card
<input type="radio" name="acctNo" value="455712341235">Platinum Card
```

Eğer değer alınır ve direk olarak SQL sorgusunda kullanılırsa, ilginç bir tür sql enjeksiyonu oluşur: yetkilendirme bilgisi manipülasyonunun ürettiği bilgi açığa çıkarma. Bağlantı havuzu tek bir kullanıcı ile veritabanına bağlandığı için, eğer SQL cümleciği şu şekilde ise diğer kullanıcıların hesaplarını görmek mümkün olabilir:

```
String acctNo = getParameter('acctNo');
String sql = "SELECT acctBal FROM accounts WHERE acctNo = '?'";
PreparedStatement st = conn.prepareStatement(sql);
st.setString(1, acctNo);
ResultSet rs = st.executeQuery();
```

Bu kod, hesap numaralarının index kullanarak elde edileceği şekilde ve diğer hesap numaralarının dışarı gözükmemesi için müşterinin biricik ID'sini içerecek şekilde tekrar yazılmalıdır:

```
String acctNo = acct.getCardNumber(getParameter('acctIndex'));

String sql = "SELECT acctBal FROM accounts WHERE acct_id = '?' AND
acctNo = '?'";
PreparedStatement st = conn.prepareStatement(sql);
st.setString(1, acct.getID());
st.setString(2, acctNo);
ResultSet rs = st.executeQuery();
```

Bu yaklaşım girdi değerlerinin 1'den x'e kadar olmasını gerektirir ve, hesapların bir Collection'da tutulduğu kabul edilirse, logic:iterate kullanılarak tek tek kullanılabilirler:

```
<logic:iterate id="loopVar" name="MyForm" property="values">
  <html:radio property="acctIndex" idName="loopVar"
value="value"/> 
  <bean:write name="loopVar" property="name"/><br />
</logic:iterate>
```

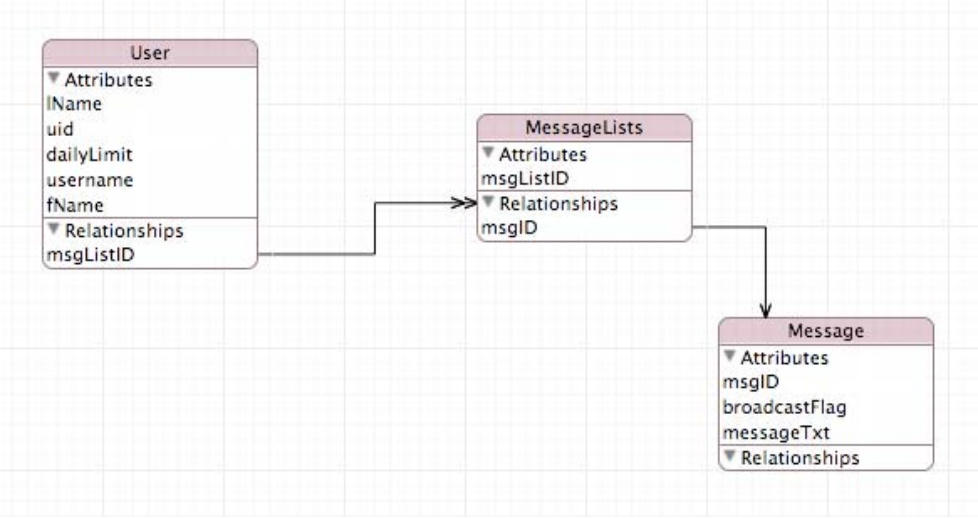
Kod, her collection'ının içeriği kadar "1" .. "x" değeleri üretecektir:

```
<input type="radio" name="acctIndex" value="1" />Gold Credit Card
<input type="radio" name="acctIndex" value="2" />Platinum Credit Card
```

Bu yaklaşım, radio buttons, checkboxes, and particularly select / option lists üretecek her girdi tipi için kullanılmalıdır.

Kullanıcıya Ait Veriler

Tamamen normalize edilmiş veritabanlarında, amaç tekrarlanan verilerin sayısını minimize etmektir. Ancak, bazı veriler dinamik olarak üretilir. Mesela, kullanıcılar bir mesaj tablosunda tutulan mesajları görebilirler. Bazı mesajlar kullanıcıya özeldir. Ancak, tamamen normalize edilmiş bir veritabanında mesaj ID'lerinin listesi başka bir tabloda tutulur:



Eğer bir kullanıcı bir mesajı silmek için işaretlerse, genellikle izlenilen yol kullanıcıdan mesaj ID'sini alıp, mesajı silmektir:

```
DELETE FROM message WHERE msgid='frmMsgId'
```

Ancak, kullanıcının belirtilen mesaj ID'yi silmek için yetkisi olduğunu nasıl anlarız? Bu tablolar kullanıcı ID'sini içerecek şekilde denormalize edilmeli veya güvenli bir şekilde mesajı silmek için tek sorgu kullanmayı sağlayacak şekilde işlemi kolaylaştırmalıdır. Mesela, opsiyonel uid sutünü ekleyerek, silme işlemi daha güvenilir hale getirilir:

```
DELETE FROM message WHERE uid='session.myUserID' and msgid='frmMsgId';
```

Verinin potansiyel olarak özel veya herkese açık kaynak olma durumunda, ek önlemler alınarak, yetkilendirme olmaksızın kullanıcıların herkese açık kaynakları silmesi engellenmelidir. Bu rol tabanlı kontroller yapılarak veya mesaj tipini ayırt edebilen SQL cümlecikleri yazılarak yapılabilir.

```
DELETE FROM message
WHERE
uid='session.myUserID' AND
msgid='frmMsgId' AND
broadcastFlag = false;
```

URL kodlama

URL içinde gönderilen veride, ki kesinlikle önerilmez, URL kodlama / kod çözme (encode / decode) işlemleri yapılmış olmalıdır. Bu XSS gibi saldırıların çalışma riskine düşürecektir.

Genel olarak, navigasyonel amaçlar dışında veri gönderimi için GET isteklerini kullanmayın.

HTML kodlama

Kullanıcıya gönderilen veri, kullanıcının görüntülemesi için güvenli olmalıdır. Bu <bean:write ...> ve buna benzer yapılar ile sağlanabilir. <bean:write...> ve benzerlerinde kullanılmadıkça <%=var%> kullanmayın.

HTML kodlama belli bir aralıktaki karakterleri HTML değerlerine değiştirir. Örnek olarak, > > halini alır. Bu kullanıcı tarayıcısında > olarak gözükür ve güvenli bir alternatiftir.

Kodlanmış dizgiler (strings)

Bazı dizgiler kodlanmış formda alınabilir. Web sunucunun ve uygulama sunucunun ilk kullanımdan önce tek seviye kanonikalizasyon sağlaması için kullanıcıya doğru lokalizasyonu göndermek çok önemlidir.

getReader() veya getInputStream() metotlarını kullanmayın, çünkü bunlar kodlanmış dizgileri çözmezler. Bu yapıları kullanmanız gerekirse, veriyi kendiniz de kanonikalizasyon yapmanız gerekir.

Ayraçlar ve özel karakterler

Bir çok programda farklı anlamlara gelebilecek bir çok karakter vardır. Eğer sadece beklediğiniz karakterleri almanız gerektiği tavsiyesini dinlediyseniz, sadece bir kaç ayraç karakteri sizi sıkıntıya sokabilir.

İşte en önemlileri:

- NULL (zero) %00
- LF - ANSI chr(10) "\r"
- CR - ANSI chr(13) "\n"
- CRLF - "\n\r"
- CR - EBCDIC 0x0f
- Quotes " '
- Commas, slashes spaces and tabs and other white space - used in CSV, tab delimited output, and other specialist formats
- <> - XML ve HTML işaretleyiciler (tag markers), yönlendirme karakterleri
- ; & - Unix ve NT dosya sistemi devamı
- @ - e-posta adreslerinde
- 0xff
- ... daha fazlası

Belirli bir teknoloji altında kodlama yapıyorsanız, hangi karakterlerin özel olduğunu belirlemeli ve girdilerde bulunmalarını önlemeli veya yeterince kaçırma (escape) işlemlerini uygulamalısınız.

İleri Okuma

- ASP.NET 2.0 Viewstate
  http://channel9.msdn.com/wiki/default.aspx/Channel9.HowToConfigureTheMachineKeyInASPNET2

Dökümanın Aslı : OWASP GUIDE 2.0.1

Çeviri **:** Bedirhan Urgun – urgunb@hotmail.com