ÖN BİLGİ

Bir makale internetten bulunup şu adreste yedeklenmiştir.

- https://www.includekarabuk.com/kitaplik/indirmeDeposu/Siber_Guvenlik_Teknik_Makaleler/Teori/BaskalarinaAitMakaleler/OWASP%20Top%2010%20(2007).pdf

Bu belge ise yedeklenen bu makaleye çalışılarak elde edilen notlarımı kapsamaktadır. Bu çıkarılan notlar belgemde alıntılar ve/veya kişisel ilavelerim mevcuttur.

1)

Hizmet dışı bırakma (DOS), herhangi bir dilde yazılmış siteyi etkileyebilecek ciddi bir saldırdır. MITRE verilerine göre DOS bu sene İlk 10 sıralamasına girmek için yeterli değildir. DOS ile ilgileniyorsanız OWASP sitesinde bulunan makale ve deneme yönergelerini inceleyebilirsiniz:

- http://www.owasp.org/index.php/Category:Denial_of_Service_Attack
- http://www.owasp.org/index.php/Testing_for_Denial_of_Service

(Page 8)

2)

Güvenli olmayan yapılandırma yönetimi, başta PHP olmak üzere tüm sistemleri etkiler. Fakat MITRE'deki sıralaması bu sene bu konuya değinmemize izin vermiyor. Uygulamanızı sunucu üzerine koyarken güvenli yapılandırma yönetimi ve denemesi hakkında OWASP sitesinde yazılmış olan makaleleri inceleyin:

- http://www.owasp.org/index.php/Configuration
- http://www.owasp.org/index.php/Testing_for_infrastructure_configuration_management

(Page 8)

3)

XSS

Siteler Arası Betik Yazma (Cross Site Scripting), çoğunlukla XSS olarak bilinen, HTML enjeksiyon yönteminin bir alt kümesidir. XSS, çok yaygın ve zararlı bir web uygulaması güvenlik sorunudur. XSS, saldırgana kurbanın tarayıcısında betik çalıştırmaya izin verip, kullanıcının oturum bilgilerini çalabilir, web sitesine zarar verebilir, saldırgana ait içerikli siteye girilebilir.

Siteler Arası Betik Yazma'nın bilinen üç türü bulunmaktadır: Yansıyan, Depolanmış ve DOM enjeksiyonu. Yansıyan XSS, en basit yapıda gerçeklenmiş açıktır (exploit). Burada kullanıcı veri kaynakları bir web sayfasıyla doğrudan yansıtılacaktır:

```
echo $_REQUEST['userinput'];
```

Depolanmış XSS, saldırganın verilerini alarak, bir dosyanın içerisine, bir veritabanına veya herhangi bir diğer arka planda çalışan sisteme yazarak, bir sonraki aşamada bu bilgileri filtrelemeksizin kullanıcılara göstermesi sonucunda ortaya çıkar. Bu CMS'ler, bloglar ve forumlar için son derece tehlikelidir. Bu yöntemle çok sayıda kullanıcının bilgileri saldırganlar tarafından görülebilir.

DOM tabanlı XSS saldırıları ise sitelerin Javascript kod ve değişkenlerinin değiştirilmesi ile ortaya çıkar.

Alternatif olarak saldırgan üç XSS tipini melez (harmanlanmış) olarak da kullanabilir. Tehlike, XSS'in tipinden değil, XSS'den kaynaklıdır. Ancak bazı durumlar daha tehlikeli olabilir. XSS, standart olmayan veya beklenilmedik web tarayıcı davranışlarının zekice yönlendirilmesi ile ortaya çıkar. XSS ayrıca web tarayıcının kullandığı bileşenlere de karşılıklı olarak ulaşma imkanı verir.

Saldırılarda çoğunlukla çok güçlü olan betik dili olan Javascript kullanılır. Javascript'in kullanılması, saldırgana sayfanın görünümü değiştirmesine, yeni bir öğe eklenmesine (Örneğin saldırganın sitesine kullanıcı bilgileri gönderecek bir oturum açma elemanı eklenmesine) olanak verir. Dahili DOM ağaç yapısı kullanılarak sayfanın doku ve görünümü değiştirilebilir veya silinebilir. Javascript XmlHttpRequest komutunun kullanımına izin verir ve tipik olarak bu komut AJAX teknolojilerinde kullanılmasına rağmen, günümüzde AJAX kullanmayan siteler de hedef durumundadır.

XmlHttpRequest kullanımıyla, bazen bir web tarayıcının izin verdiği kaynaklarının etrafından dolaşması sağlanarak; kurbanın verileri saldırganın sitesine yönlendirilebilir ve karmaşık worm ve zararlı zombi kodları oluşturularak web tarayıcısı çalıştığı sürece kullanılabilir. AJAX saldırıları, tehlikeli "cross site request forgery" (CSRF) (Siteler Ötesi İstek Sahteciliği) yapmak için görünür olmak zorunda değildir veya kullanıcı etkileşimi gerektirmemektedir.

(Page 10-11)

XSS’ten Korunma

• **Gelen verilerin doğrulanması :** Uygulamada kullanılabilecek bütün parametreler doğrulanmalıdır. Bunun için, öncelikle gelen verilerin uzunluğu, tipi, söz dizimi ve yerine göre kullanım kurallarını kontrol edildikten sonra verinin görüntülenebileceği veya kayıt edilebileceği, standart bir doğrulama mekanizması kullanılır. Bunun için "Bilginin iyi niyetli olduğu kabul edildi" doğrulama stratejisi kullanılmalıdır. Potansiyel kötü niyetli veriler reddedilerek, sistemden sterilize edilmelidir. Unutulmaması gereken bir nokta validation bildirimlerinin içeriği saldırganın işine yarayacak bilgi içermemelidir.

• **Kara liste araçları kullanmayın :** Kodlanmış çıktıların, sisteme girişindeki XSS'leri saptamak için kara liste kullanmayın. Yanlızca birkaç karakterden oluşan ("<" ">" ve diğer benzer karakter öbekleri veya betik gibi) kodları kontrol ederek silen yapılar zayıf ve saldırılmaya uygundur. Örneğin böyle bir uygulamada çift"<b>" etiketi kontrol edilemez ve bu durum benzer içerikler içinde geçerlidir. XSS çok sayıda sürpriz yöntemler içereceğinden kolaylıkla kara liste kontrolü geçilebilir.

(Page 12)

Kullanılan dil için belirli öneriler:

● **PHP: Çıktı verilerine güvenmemek:** htmlentities() veya htmlspecialchars() veya tercihen OWASP PHP Anti-XSS kütüphanesini kullanın. Henüz kapatmamışsanız register_globals'ı kapatın.

(Page 13)

(!) Java ve .NET not alınmadı. Dilersen dökümandan okuyabilirsin.

4)

Injection
Enjeksiyon açıkları, özellikle SQL enjeksiyon, web uygulamalarında ortak bir açıktır. Çok fazla tipte enjeksiyon tipi vardır: SQL, LDAP, Xpath, XSLT, HTML, XML, OS komutları ile enjeksiyon ve daha fazlası. Enjeksiyon, kullanıcı kaynaklı veri içerisinde bir komut veya sorgunun bir bölümünün yorumlayıcıya gönderilmesiyle gerçekleştirilir. Saldırganlar, ustalıkla planlanmış komutlar aracılığıyla, yorumlayıcıya çalışabilir komutlar göndererek özel veri alanlarına erişim sağlarlar. Enjeksiyon açıkları, uygulama üzerinden saldırganların isteğine bağlı olarak, oluşturma, okuma, güncelleştirme veya silme izinlerinden herhangi birini verir.

(Page 15)

Injection'dan Korunma
● **En düşük ayrıcalık verilmeli.** Veritabanına ve diğer arka plan uygulamalarına bağlanıldığında en düşük imtiyaz ile bağlanılmalıdır.

● **Detaylı hata mesajlarından kaçınılmalı.** Saldırganın kullanabileceği detaylı hata mesajlarından kaçınılmalıdır.

● **Dinamik sorgu arayüzlerini kullanmayın.** (Örneğin mysql_query() veya benzerleri...)

● **Basit kaçış fonksiyonlarını kullanmayın.** Örneğin PHP'nin addslashes() veya yerine koyma fonksiyonu str_replace("'", "''"). Bu fonksiyonlar saldırganlar için zayıflık oluşturacaktır. PHP için, MySQL kullanılıyorsa mysql_real_escape_string() kullanılır veya tercihan PDO kullanılabilir, böylece escape gerektirmez.

(Page 16)

Kullanılan dil için belirli öneriler:
● PHP-PDO ile güçlü tipteki sorgular kullanılmalı (bindParam() kullanılabilir.)

(Page 17)

5)

Zararlı Dosya Çalıştırma (Shell Dosyası Atma)
Zararlı dosya saldırgana aşağıda belirtilenleri yapma izni verir:

- Uzaktan kod çalıştırma
- Uzaktan rootkit kurma ve sistemin tamamını ele geçirme
- Windows üzerinde, PHP'nin SMB dosyası kullanılarak dahili sistem ele geçirilebilir.

(Page 18)

Zararlı Dosya Çalıştırmadan Korunma
● **Kullanıcı girişlerini kontrol edecek güçlü mekanizmalar kullanın** ve gelen verinin "iyi olduğu kabul edildi" stratejisi oluşturun.

(Page 20)

6)

Emniyetsiz Doğrudan Nesne Referansı
Bir doğrudan nesne referansı, geliştiricinin, URL olarak veya parametre yoluyla, dosya, dizin, veritabanı kaydı veya anahtar gibi, iç uygulama nesnesine ait bir referansı açığa çıkardığı zaman meydana gelir. Saldırgan, eğer erişim kontrol denetimi yoksa, izinsiz diğer nesnelere erişmek için doğrudan nesne referanslarını kullanabilir.

Örneğin, İnternet bankacılık uygulamalarında, birincil anahtar (primary key) olarak hesap numarasını kullanmak oldukça yaygındır. Bu sebeple İnternet arayüzünde doğrudan hesap numarası kullanımı mevcuttur. Geliştiriciler, SQL saldırılarını engellemek için parametreli sorgulama kullanmış olsalar dahi eğer hesabı görmeye yetkili kullanıcı veya hesap sahibi için hiçbir fazladan denetim yoksa, hesap numarası parametresiyle oynayan saldırgan, bütün hesapları görebilir veya değiştirebilir.

Örneğin, eğer tasarlanan kod dosya adlarını veya yollarını belirtmesi için kullanıcı girişine izin veriyorsa, saldırgan uygulamanın çalışma dizininden dışarı ulaşabilir ve diğer kaynaklara erişmesi mümkün olur.

```
<select name="language"><option value="tr">Türkçe</option></select>
...
require_once ($_REQUEST['language'] . "lang.php");
```

Böyle bir koda, İnternet sunucusunun dosya sistemindeki herhangi bir dosyaya erişmek için "../../../../etc/passwd%00" gibi bir dizgi kullanarak, null byte injection yöntemiyle (boş byte saldırısı ile) saldırabiliriz. %00 ifadesi sonradan gelecek lang.php string'ini elimine edecektir. Çünkü string okuması null gelene kadar devam eder. Biz %00 ile null'ı erken getirdiğimiz için sunucu string'i okumayı bırakacaktır ve devamındaki); kodunu okuyup yoluna devam edecektir.

(Page 23)

Korunma Yöntemi

Doğrudan nesne referanslarını açığa vurmaktan kaçınılmalıdır. Eğer doğrudan nesne referansı kullanılması şart ise parametrenin referansı kullanmadan önce kimlik doğrulamadan geçtiğinden emin olunmalıdır.
(Page 24)

7)

Cross Site Request Forgery
Kurban tarafından görülen aşağıdaki tag'a sahip herhangi bir İnternet sayfası oturum kapatma isteği oluşturacaktır:

```
<img src="http://www.example.com/logout.php">
```

Eğer İnternet hizmeti veren bir banka, gelen istekleri işlemek için uygulamaya izin verseydi, örneğin para transferi gibi, benzer bir saldırıya izin vermiş olurdu:

```
<img src="http://www.example.com/transfer.do?frmAcct=document.form.frmAcct
&toAcct=4345754&toSWIFTid=434343&amt=3434.43">
```

Korunma Yöntemi

● URL ve her form için tarayıcı tarafından otomatik olarak istenmeyecek rastgele işaretler eklenmelidir. Örneğin,

```
<form action="/transfer.do" method="post">
        <input type="hidden" name="8438927730" value="43847384383">
        ...
</form>
```

● Önemli işler veya hassas veriler için GET isteği (URLs) kullanılmamalıdır. Kullanıcı tarafından hassas verinin işleme tabi tutulduğu zamanlarda sadece POST metodu kullanılmalıdır. Yine de, benzersiz bir URL'i yaratmak için rastgele işaret içeren URL, CSRF'i gerçekleştirmeyi neredeyse imkansız kılar.

(Page 26-27)

8)

Bütün kimlik bilgileri, kriptografik özet veya şifrelenen formda depolanmalıdır. (Page 31)

Her sayfada bir oturumu kapatma bağlantısı olduğu garanti edilmelidir. Oturum kapatma işlemi, bütün sunucu tarafı oturum durumunu ve kullanıcı tarafı çerezlerini yok etmelidir. İnsan faktörler düşünülmeli: Doğrulama için soru sorulmamalı çünkü kullanıcılar, başarılı bir şekilde oturumu kapatmak yerine, pencere veya sekme kapatmayı seçecektir. (Page 32)

Korunan verinin değerine göre etkin olmayan bir oturumdan otomatik olarak dışarı çıkartan bir zaman aşımı periyotu kullanılmalı. (Kısa periyot her zaman daha iyidir) (Page 32)

Kullanıcı, şifresini değiştireceği zaman, eski şifresi kontrol edilmelidir. (Page 32)

9)

Güvensiz Kriptografik Depolama

Yaygınca görülen şifreleme sorunları şunlardır:

- Güçlü algoritmaları güvensiz bir şekilde kullanmak
- Zayıflığı ispatlanmış algoritmaları kullanmaya devam etmek (MD5, SHA-1, RC3, RC4, etc…)

Öneriler

- **Zayıf algoritmalar kullanmayın.** MD5/SHA-1 gibi zayıf algoritmalar kullanmayın. Daha güvenli alternatifleri olarak SHA-256 yada daha iyisi kullanın.

10)

URL Erişimini Kısıtlamada Bozukluk

Bu zayıflık için ilk saldırı yöntemi “forced browsing” olarak adlandırılmaktadır. Korunmasız sayfaları bulmak için birer birer deneme (brute force) atakları ya da linkleri tahmin etme yöntemleri uygulanabilir.

Sunum katmanındaki yada uygulamadaki tüm URL’lerin içinde erişim kontrolü onayı güçlendirilmelidir.

(Page 40)

Korunma

- Business logic korunmasız bırakılıp sunum katmanı içine erişim kontrolü koymak yeterli değildir.

- **İlerde kullanılmayacak bütün dosya türlerine ve uzantılarına erişimi bloklayın.** İdeal olarak, kullanmaya niyetli olduğunuz .html, .php, .pdf gibi uzantılara sadece izin vererek bilinen iyilere izin ver “accept known good” politikasını izleyebilirsiniz. Bu uygulama sayesinde herhangi bir şekilde log dosyalarına, xml dosyalarına, vb istemediğiniz direct erişim girişimini bloklamış olursunuz.

- **Virüs korumanızı güncel tutun ve güvenlik yamalarınızı zamanında yaparak,** kullanıcı uygulama verilerini elinde tutan XML işlemci, kelime işlemci, resim işleyici programlarınızı güncel tutunuz.

(Page 41)

11)

Eğer eğitim için bütçeniz varsa **güvenli kod yazma için eğitim arayın.** Eğer eğitim için ayrılmış bütçeniz yoksa yöneticilerinizle bu konuyu görüşün.
Geleneksel olmayan herhangi bir dizayn uygulaması iyi dozda bir güvenlik gerektirir.
Standart kodlamayı benimseyin. Bu sizi daha güvenli kodlamaya yönlendirir.
Kodunuzu güvenlik sorunlarına karşı test edin. Bunu kendinizde alışkanlık haline getirin.

Açık kaynak, web uygulama güvenliği için bir nevi meydan okumadır. Tek kişi tarafından geliştirilen kişisel projelerden, Apache, Tomcat gibi büyük projelere, PostNuke gibi büyük ölçekli web uygulamalarına kadar milyonlarca açık kaynak projesi bulunmaktadır.