OWASP ILK 10



EN KRİTİK 10 WEB UYGULAMASI GÜVENLİK ZAYIFLIKLARI

2007 GÜNCELLEMESİ

© 2002-2007 OWASP Kuruluşu

Bu döküman <u>Attribution-ShareAlike 2.5</u> lisansı kapsamında oluşturulmuştur.



ÇEVİRİ

Bu çeviri, OWASP Türkiye ile koordineli olarak, ULAK-CSIRT (http://csirt.ulakbim.gov.tr) Web Güvenliği Çalışma Grubu tarafından gerçekleştirilmiştir.

Çeviri öneri ve düzeltmelerinizi enis.karaarslan@ege.edu.tr adresine gönderebilirsiniz.

Enis Karaarslan, Ege Üniversitesi

Murat Sürücü, Karaelmas Üniversitesi

Nazım Karadağ, Ondokuz Mayıs Üniversitesi

Oğuzhan Yalçın, Gazi Üniversitesi

Vedat Fetah, Ege Üniversitesi



ULAK-CSIRT (http://csirt.ulakbim.gov.tr)

Web Güvenliği Çalışma Grubu, 2007

İÇİNDEKİLER

İÇİNDEKİLER	3
TANITIM	4
ÖZET	6.
METODOLOJİ	7.
A 1 - SİTELER ARASI BETİK YAZMA (XSS)	11
A2- ENJEKSİYON AÇIKLARI	1.5
A3- ZARARLI DOSYA ÇALIŞTIRMA	1.8
A4- EMNİYETSİZ DOĞRUDAN NESNE REFERANSI	23
A5- SİTELER ÖTESİ İSTEK SAHTECİLİĞİ (CSRF)	26
A6- BİLGİ SIZINTISI VE UYGUNSUZ HATA IŞLEME	29
A7- IHLAL EDILMIŞ KİMLİK DOĞRULAMA VE OTURUM YÖNETİMİ	31
A8- GÜVENSIZ KRİPTOGRAFİK DEPOLAMA	34
A9- GÜVENSIZ İLETİŞİMLER	37
A 10- URL ERİŞİMİNİ KISITLAMADA BOZUKLUK	40
BURADAN NEREYE DEVAM ETMELİ	43
REFERANSLAR	46
MİNİ SÖZLÜK	48



TANITIM

OWASP İlk 10 2007'ye hoş geldiniz. Tümüyle yeniden yazılmış bu basım, en önemli web uygulaması zayıflıklarını, bunlara karşı nasıl tedbir alınacağını ve daha fazla bilgiye erişmek için bağlantıları içermektedir.

HEDEF

OWASP İlk 10 2007'nin birincil amacı; geliştiricileri, tasarımcıları, yazılım mimarlarını ve kurumları en çok rastlanılan web uygulaması güvenlik zayıflıklarının önemi konusunda eğitmektir. İlk 10, bu zayıflıklara karşı korunmanın basit yöntemlerini içermektedir, "Güvenli kodlama" güvenlik programına başlamanız için mükemmel bir noktadır.

Güvenlik bir süreçtir (Güvenlik tek anlık bir olay değildir). Kodunuzu bir seferde güvenli hale getirmek mümkün değildir. 2008 yılı itibari ile bu İlk 10 listesi değişmiş olacak, kodunuzda tek satır bile bir değişiklik yapmadan tüm güvenlik açıklarına karşı korunmanız mümkün olmayacak ve güvenlik zayıflıklarınız olacaktır. Daha fazla bilgi için <u>Buradan nereye devam etmeli</u> bölümündeki önerileri gözden geçirin.

Bir güvenli kodlama birimi program yaşam döngüsünün tüm basamakları ile ilgilenmelidir.

Güvenli web uygulamalarının geliştirilmesi sadece güvenli bir SDLC (Yazılım geliştirme yaşam döngüsü) kullanıldığında mümkündür. Güvenli yazılımlar tasarım sürecinde, geliştirme sürecinde ve çıktığı anda güvenlidir. Bir web uygulamasının güvenliğini etkileyen en az 300 etken mevcuttur. Bu 300+ etken her web uygulama geliştiricisinin okuması gereken OWASP Klavuzu'nda belirtilmiştir.

Bu doküman bir standart değil, her şeyden önce bir eğitim dokümanıdır. Lütfen bu dokümanı bizimle iletişime geçmeden bir politika ya da standart olarak benimsemeyin. Eğer bir güvenli kodlama politikasına ya da standardına ihtiyacınız varsa, OWASP tarafından yürütülmekte olan güvenli kodlama politikaları ve standartları belirleme projeleri bulunmaktadır. Lütfen bunlara katılın ya da bu çalışmaları maddi olarak destekleyin.

TEŞEKKÜRLER

MITRE'ye CVE verisindeki Zayıflık Tip Dağılımlarını serbest kullanıma açtığı için teşekkürler. OWASP Top Ten projesi <u>Aspect Security</u>'nin önderliği ve sponsorluğunda yürütülmüştür.



Proje Yöneticisi: Andrew van der Stock (Yürütücü Direktör, OWASP Kuruluşu)

Alt Yürütücüler: Jeff Williams (Üye, OWASP Kuruluşu), Dave Wichers (Konferans Üyesi, OWASP Kuruluşu)

Denetmenlerimize teşekkür etmek istiyoruz:

 Raoul Endres'e Top Ten projesinin tekrar yürütülmesine yardımcı olduğu için ve değerli yorumları için

- Steve Christey'e (MITRE) geniş kapsamlı ve önemli değerlendirmeleri ve MITRE verilerini eklediği için
- Jeremiah Grossman'a (White Hat Security) önemli değerlendirmeleri ve otomatik algılama sistemlerinin başarısı(yada tam tersi) hakkında eklediği bilgiler için
- Sylvan von Stuppe'ye önemli değerlendirmeleri için
- Colin Wong, Nigel Evans, Andre Gironda, Neil Smithline e-posta ile ulaştırdıkları yorumları için



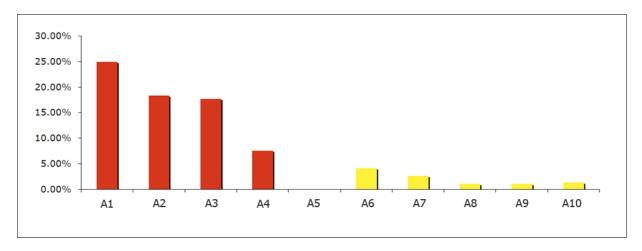
Ö7FT

A1 – Siteler Arası Betik Yazma (XSS)	XSS açıkları uygulama kullanıcıdan veri alıp, bunları herhangi bir kodlama ya da doğrulama işlemine tabi tutmadan sayfaya göndermesi ile oluşur. XSS saldırganın kurbanın tarayıcısında kullanıcı oturumları bilgilerin çalınmasına, web sitesinin tahrif edilmesine veya solucan yüklenmesine sebep olan betik çalıştırmasına izin verir.
A2 – Enjeksiyon Tasarım Hatası	Enjeksiyon saldırılarına , özellikle SQL enjeksiyonu, web sitelerinde sıkça rastlanmaktadır . Enjeksiyon, kullanıcı tarafından alınan verinin yorumlayıcıya (interpreter) komut ya da sorgunun bir parçası olarak gönderilmesi durumda oluşur. Saldırganın düşmanca gönderdiği veriler yorumlayıcının istenmeyen komutları çalıştırmasına veya veriyi değiştirmesine sebep olur.
A3 – Zararlı Dosya Çalıştırma	Uzaktan Dosya Ekleme (RFI) ataklarına karşı zayıflıkları olan kod, saldırganın sunucunun güvenliğini tehlikeye düşürebilecek kadar tehlikeli düşmanca kodlar ve veriler yüklemesine yol açar. Kötü amaçlı dosya çalıştırılması kullanıcıdan dosya adı veya dosya kabul eden PHP, XML veya herhangi bir çerçeveyi (framework) etkileyebilir.
A4 – Güvensiz Doğrudan Nesne Başvurusu	Doğrudan Nesne Referanslama; geliştirici dosya, dizin, veritabanı kaydı gibi bir bilgiyi URL veya form parametresi olarak alıp bunu uygulamaya referans olarak tanımladığı zaman oluşur. Böylece saldırgan referansı manipule ederek yetkisi olmayan nesnelere erişebilir.
A5 — Siteler Ötesi İstek Sahteciliği (CSRF)	CSRF saldırıları; sisteme girişi yapmış mağdura ait tarayıcının, bir web uygulamasına sonradan saldırganın yararına olacak, düşmanca hazırlanmış ve önceden doğrulanmış bir istek göndermesine sebep olur. CSRF, saldırdığı web uygulaması kadar güçlü olabilir.
A6 – Bilgi Sızıntısı ve Uygunsuz Hata İşleme	Uygulamalar istemeden de olsa yapılandırmaları, iç işleyişleri hakkında bilgi sızdırabilir veya uygulama sorunlarından dolayı güvenlik ihlallerine yol açabilir. Saldırganlar bu zayıflıkları daha ciddi saldırılar gerçekleştirmek veya önemli bilgileri çalmak için kullanabilir.
A7 – İhlal Edilmiş Kimlik Doğrulama ve Oturum Yönetimi	Hesap Bilgileri ve oturum anahtarları çoğu zaman düzgün olarak korunmamaktadır. Saldırganlar şifreleri ve kimlik denetimi anahtarlarını kullanıcının diğer bilgilerini elde etmek için kullanabilirler.
A8 – Güvensiz Kriptografik Depolama	Web uygulamaları verilerin ve bilgilerin güvenliğini sağlamak için nadiren kriptografik fonksiyonları kullanırlar. Saldırganlar zayıfça korunan veriyi, kimlik hırsızlığı ve kredi kartı dolandırıcılığı gibi diğer suçları işlemek için kullanırlar.
A9 – Güvensiz İletişimler	Uygulamalar önemli iletişim bilgilerini ve veri alış verişlerini korumaları gerekirken, çoğu zaman ağ trafiğini şifrelemekte yetersiz kalmaktadırlar.
A10 – URL Erişimini Kısıtlamada Bozukluk	Çoğu zaman uygulamalar, bağlantı ve URL satırlarını yetkili olmayan kullanıcılara sadece göstermeyerek hassas işlevselliği korumaktadırlar. Saldırganlar bu URL'lere doğrudan erişerek, yetkisi olmayan işlemleri gerçekleştirmek için bu zayıflığı kullanabilirler.

Tablo 1: 2007 için İlk 10 Web uygulama açıkları

METODOLOJİ

İlk 10 2007 için metodolojimiz basitti: 2006 için MITRE Zayıflık Eğilimlerini aldık ve İlk 10 web uygulama güvenlik sorununu tespit ettik. Dereceye giren sonuçlara ait grafik aşağıdadır.



Figür 2: 2006 için İlk 10 web uygulama açıkları için MITRE verisi

MITRE ham açıklar verisi ile bölüm başlıklarının birebir uyuşmasına özen göstermiş olsak da, kasıtlı olarak son bölümleri temel etkenleri işaret etmesi için değiştirdik. MITRE'nin 2006 ham verileri ile ilgileniyorsanız, OWASP İlk 10 web sitesinde verileri içeren bir Excel çalışma sayfası bulunmaktadır.

Bütün korunma önerileri en yaygın 3 web uygulama altyapısı olan Java EE, ASP.NET ve PHP için çözümler içermektedir. Ruby on Rails veya Perl gibi altyapılarda ise önerilerimiz belirli gereksinimlere kolayca adapte edilebilecek yapıdadır.

NEDEN BAZI ÖNEMLİ SORUNLARI LİSTEDEN ÇIKARTTIK

Kontrol Edilmemiş değer, her geliştirme ekibi için zorlu bir uğraştır ve çoğu uygulama güvenlik açığının temelinde yer alır. Nitekim diğer konuların büyük çoğunluğu çözümün bir parçası olarak değerlerin kontrolünü önermektedir. Halen sıkı bir biçimde web uygulamalarının bir parçası olarak merkezi bir değer kontrol mekanizmasının geliştirilmesini önermekteyiz. Daha fazla bilgi için, aşağıdaki linkleri takip ederek OWASP sitesinde bulunan değer kontrol makalelerini okuyunuz:

- http://www.owasp.org/index.php/Data Validation
- http://www.owasp.org/index.php/Testing for Data Validation

Arabellek taşmaları, tam sayı taşmaları ve metin biçimleme sorunları, C ve C++ dillerinde yazılmış uygulamalar için çok ciddi zayıflıklardır. Bu sorunlar için çözümler, web uygulaması dışı konularda oluşturulmuş SANS ve CERT gibi güvenlik toplulukları ve programlama dili araç geliştiricileri tarafından ele alınmıştır. Eğer kodunuz bu sorunlara sebep olabilecek bir dilde yazılmış ise, size OWASP'da bulunan aşağıdaki makaleleri okumanızı tavsiye ediyoruz:

- http://www.owasp.org/index.php/Buffer_overflow
- http://www.owasp.org/index.php/Testing for Buffer Overflow



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

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

NEDEN BAZI ÖNEMLİ KONULARI EKLEDİK

Siteler Ötesi İstek Sahteciliği (CSRF), OWASP İlk 10'un bu sürümüne eklenen en büyük yeniliktir. Ham verilerde 36. sırada görünmesine rağmen, biz uygulamaların, özellikle de önemli verilerle uğraşan ve değerli uygulamaların, korunma girişimlerine bugünden başlaması gerektiğine inanıyoruz. CSRF sıralamasının belirttiğinden daha yaygın ve çok tehlikeli bir açıktır.

Kritografinin güvenlik kaygısı gütmeksizin kullanılması, ham MITRE verisine göre 8. ve 9. sıradaki güvenlik açıklarının sebebi değildir, fakat birçok bilgi sızması ve uyumluluk sorunun temel etkenidir. (Özellikle PCI DSS 1.1 uyumluluğu ile beraber)

ATAKLAR DEĞİL ZAYIFLIKLAR

İlk 10'un bir önceki versiyonu; zayıflıklar, ataklar ve korunma yöntemlerinin bir karışımını içeriyordu. Bu sefer tümüyle zayıflıklar üzerine yoğunlaştık fakat kullanılan terminoloji çoğu zaman atak ve zayıflığı birleştiriyor. Eğer kuruluşlar, bu dokümanı uygulamalarının güvenliğini arttırmak ve riskleri azaltmak için kullanırlarsa büyük olasılıkla aşağıdakilerde azalma yaşayacaklardır:

- Parola Balıkçılığı (Phishing) saldırıları bu zayıflıklardan herhangi birini tetikleyebilir. Özellikle
 XSS ve zayıf güvenlik kontrolleri ya da kimlik doğrulama işlemleri (A1, A4, A7, A10)
- Gizlilik ihlalleri zayıf kontrol, iş süreçleri ve zayıf kimlik doğrulama işlemleri (A2, A4, A6, A7, A10)
- **Kimlik hırsızlığı** zayıf ya da bulunmayan kriptografik kontroller (A8 ve A9), uzak dosya dahil etme (A3) zayıf güvenlik kontrolleri ya da kimlik doğrulama işlemleri (A4, A7, A10)
- Sistem uyuşmazlığı, veri değiştirme veya veri bozma saldırıları Enjeksiyon (A2) ve uzak dosya dahil etme ile (A3)
- Finansal kayıp yetkisiz hareketler ve CSRF atakları ile (A4, A5, A7, A10)
- İtibar kaybı yukarıdaki herhangi bir zayıflığın istismarı ile (A1 ... A10)

Bir kuruluş tepkisel (reactive) kontroller hakkında endişe duymayı bırakıp işlerinde karşılaşabilecekleri riskleri insiyatifi ele alarak (proaktive) azaltma yoluna giderse; düzenleyici

rejimlerle olan uyumluluğunu arttır, işlemsel giderleri azaltır ve sonuç olarak arzulanan çok daha sağlıklı ve güvenli bir sisteme sahip olurlar.

EĞİLİMLER

Yukarıda tanımlanan metodoloji, güvenlik araştırma topluluklarının keşiflerine göre İlk 10'un tanımlanmasına yön vermiştir. Bu keşif modeli <u>gerçek saldırı</u> metodlarıyla özellikle giriş seviyesi (script kiddy) saldırganlarla ilgili olması bakımından benzerlik gösterir. Uygulamanızı İlk 10'a karşı korumak, sık rastlanan saldırı çeşitlerine karşı bir parça da olsa güvenliğinizi sağlayacaktır; fakat daha önemlisi, uygulamanızın güvenliğini arttırmak için bir yön belirlemenize yardımcı olacaktır.



KONUMLANDIRMA

Konularda bir önceki yıla göre bile büyük değişiklikler oldu. Artık yeni zayıflıkları, saldırıları ve önlemleri karşılayacak şekilde güncellenmediği için WAS XML adlandırma şemasını kullanmıyoruz. Aşağıdaki tablo, İlk 10 2004 ve ham MITRE verilerine göre bu versiyonun konumunu belirtmektedir:

OWASP İIK 10 2007	OWASP İlk 10 2004	MITRE 2006 Ham Sıralama
A1. Siteler Arası Betik Yazma (XSS)	A4. Siteler Arası Betik Yazma (XSS)	1
A2. Enjeksiyon Saldırıları	A6. Enjeksiyon Saldırıları	2
A3. Kötü amaçlı Dosya Çalıştırılması (YENİ)		3
A4. Güvenli olmayan Direk Nesne Referanslama	A2. İhlal Edilmiş Erişim Kontrolü (2007 T10 da ayrıldı)	5
A5. Sunucu Taraflı Çapraz Kod Çalıştırma (CSRF) (YENİ)		36
A6. Bilgi Sızdırma ve Uygun olmayan Hata Kontrolü	A7. Uygun olmayan Hata Kontrolü	6
A7. İhlal Edilmiş Kimlik Doğrulama ve Oturum Yönetimi	A3. İhlal Edilmiş Kimlik Doğrulama ve Oturum Yönetimi	14
A8. Güvenliği Sağlanmamış Kriptografik Depolama	A8. Güvensiz Depolama	8
A9. Güvensiz İletişimler (YENİ)	A10 Güvensiz Yapılandırma Yönetimi konusu altında tartışıldı	8
A10. URL'ye Erişimi Engelleyememe	A2. İhlal Edilmiş Erişim Denetimi (2007 T10 da ayrıldı)	14
<2007 de kaldırıldı >	A1. Kontrol Edilmemiş Değerler	7
< 2007 de kaldırıldı>	A5. Arabellek Taşması	4, 8 ve 10
< 2007 de kaldırıldı>	A9. Servis Reddi	17
< 2007 de kaldırıldı>	A10. Güvensiz Yapılandırma Yönetimi	29

A1- SİTELER ARASI BETİK YAZMA (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, bir uygulamadaki eksiklikten kaynaklı olarak, bir kullanıcıdan web tarayıcı aracılığı ile veri alındığında, kullanıcının içeriği onaylamasından ve kodlamasından bağımsız olarak gerçekleşir.

XSS, saldırgana kurbanın tarayıcısında betik çalıştırmaya izin verip, kullanıcının oturum bilgilerini çalabilir, web sitesine zarar verebilir, saldırgana ait içerik siteye girilebilir, site parola balıkçılığı (phishing) saldırılarını iletir duruma getirilebilir ve kötü niyetli yazılım çalıştırılabilir.

Kötü niyetli betiklerde çoğunlukla JavaScript kullanılır, fakat kullanıcının web tarayıcısının desteklediği herhangi bir betik dili de saldırgan için potansiyel hedef dildir.

ETKİLENEN ORTAMLAR

Bütün web uygulama yapıları, Siteler Arası Betik Yazma için saldırıya uygun ortamlardır.

ZAYIFLIK

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 diğer kişiler 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 (A-5 bölümüne bakınız.)



GÜVENLİĞİN DOĞRULANMASI

Burada amaç, HTML sayfalarının içeriğinin şifrelenmeden veya onaylanmadan önce, uygulamada kullanılabilecek bütün parametreler doğrulanmalıdır.

Otomatik yapılan yaklaşımlar: Otomatik zayıflık araçlar, XSS enjeksiyon parametrelerini kullanarak zayıflıkları saptayabilir, ancak sürekli değişen XSS saldırılarında başarılı olamaz, özellikle XSS enjeksiyonu, Kimlik kontrolü yöntemi ile engellenir (örneğin, eğer bir kullanıcı girişleri, saldırgan kod içeriyorsa bunu ancak sistem yöneticisi görebilir). Otomatikleştirilmiş kaynak kodu tarama araçları zayıf ve tehlikeli kodları bulabilir, fakat çoğunlukla onaylanmış veya şifrelenmiş tehlikeleri saptayamaz, bunun yanında yanlış uyarılar verebilir. Hiçbir araç tipi DOM tabanlı XSS saldırılarını bulamaz, bunun anlamı yalnızca otomatik test araçları ile taranan Ajax tabanlı uygulamaların çoğunlukla risk altında olduğudur.

El ile yapılan yaklaşımlar: Eğer merkezi bir doğrulama ve kodlama mekanizması kullanıldığında, çoğunlukla kodların güvenliğini el ile kontrol etmek de bir yoldur. Eğer dağınık bir uygulama yapısı kullanılıyorsa, doğrulama çok fazla zaman alacaktır. Test süresi çok fazla zaman alacaktır, çünkü çoğu uygulama çok geniş bir saldırı yüzeyine yayılmıştır.

KORUNMA

XSS için en iyi korunma, bir kontrol listesi ile, bütün gelen verileri ve kodlayarak giden verilerin tümünü karşılaştırarak zararlı verileri bulmaktır. Kontrolden geçmeyi başaran saldırılar ve kodlanmış bölümlerden gelen betik enjeksiyonları web tarayıcıda çalışacaktır.

XSS'in tamamen önüne geçmek için yapısal tutarlılığı kontrol edecek bir uygulama gerektirir. Bu uygulamalar aşağıdaki gibidir:

- Gelen verilerin doğrulanması: 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 hata mesajlarının içeriği saldırganın işine yarayacak bilgi içermemelidir.
- Çıkan verilerin güçlü bir şekilde şifrelenmesi: Garantiye almak için, bütün kullanıcı kaynaklı veriler gönderilmeden önce (HTML veya XML'e bağlı bir çıkış mekanizmasına) uygun bir şekilde kodlanmalıdır. Kodlanan tüm verilerin dışında kalması gereken verilerde bir araç ile kontrol edilmelidir. Bunun için Microsoft Anti-XSS kütüphanesi ve yakında yayınlanacak olan OWASP PHP Anti-XSS kütüphanesi kullanılabilir. Bunun yanında her sayfanın karakter kodlamasının da önceden ayarlanması riskleri azaltacaktır.
- Belirli Çıkış verisi kodlaması: (Örneğin ISO 8859-1 veya UTF 8). Bu sayede saldırganın, kullanıcılarınız için kodlamayı seçmesine izin verilmemiş olunur.
- 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"" etiketi kontrol edilemez ve bu durum benzer içerikler içinde geçerlidir. XSS çok sayıda sürpriz yöntemler içereceğinden kolaylıkla kara liste kontrolü geçilebilir.

 Varsayılan hata çıktılarını izlenmesi: Çoğunlukla sisteme gelen verilerin kodun çözülmesinin ardından kullanılan dahili yazılım ile gösterilmeden önce kontrol edilmelidir. Elbette ki uygulamanız iki kez girilen benzer veri girişlerinin kodu çözülemeyecektir. Bu şekilde tehlikeli veri girişleri ile hatalar kullanılarak, kontrol işlemini gerçekleştiren beyaz liste şemaları atlatılabilir.

Kullanılan dil için belirli öneriler:

- Java: Veri çıkış mekanizmasını destekleyecek kod kullanmak. Örneğin <bean:write ...>, veya standart olarak kullanılan JSTL escapeXML="true" bağlı olan <c:out ... >. <%= ... %> yapısını iç içe kullanmayın (Diğer bir yandan, bu tam anlamıyla kodlanmış bir veri çıkış mekanizması olacaktır.)
- .NET: Microsoft Anti-XSS Kütüphanesinin kullanımı: MSDN'den 1.5 sürümüne rahatlıkla erişebilirsiniz. Bu kütüphanenin kullanımıyla, "Request" nesnesi ile atanmamış form alanlarındaki veriler: : username.Text = Request.QueryString("username"); bu kütüphaneyi kullanacaktır. Yani .NET kontrollerinden çıkan verileri otomatik olarak kodlanacaktır.
- PHP: Çıktı verilerine güvenmemek : htmlentities() veya htmlspecialchars() veya tercihan
 OWASP PHP Anti-XSS kütüphanesini kullanın. Henüz kapatmamışsanız register_globals'i kapatın.

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4206
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3966
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5204

REFERANSLAR

- CWE: CWE-79, Siteler Arası Betik Yazma (XSS)
- WASC Tehlike sınıflandırması : http://www.webappsec.org/projects/threat/classes/cross-site-scripting.shtml
- OWASP- Siteler Arası Betik Yazma, http://www.owasp.org/index.php/Cross_Site_Scripting
- OWASP- XSS için deneme yapmak, http://www.owasp.org/index.php/Testing for Cross site scripting
- OWASP Stinger Projesi (Java EE kontrol filtresi), http://www.owasp.org/index.php/Category:OWASP Stinger Project
- OWASP PHP Filtre Projesi http://www.owasp.org/index.php/OWASP_PHP_Filters
- OWASP Kodlama projesi http://www.owasp.org/index.php/Category:OWASP_Encoding_Project
- RSnake, XSS Aldatma Katmani, http://ha.ckers.org/xss.html
- Klein, A., DOM Tabanlı Siteler Arası Betik Yazma, http://www.webappsec.org/projects/articles/071105.shtml



• .NET Anti-XSS Kütüphanesi, http://www.microsoft.com/downloads/details.aspx?FamilyID=efb9c819-53ff-4f82-bfaf-e11625130c25&DisplayLang=en

A2- ENJEKSİYON AÇIKLARI

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. En kötü durum senaryosu, bu açıklardan yararlanan saldırganın tamamen uygulama ve önemli sistem bileşenleri ile güvenlik duvarı katmanına erişebilmesidir.

ETKİLENEN ORTAMLAR

Yorumlayıcıları kullanan bütün web uygulamaları veya enjeksiyon saldırılarına açık diğer işlemler bu saldırıdan etkilenirler. Saldırganlar, enjeksiyon yöntemi ile kullanılan uygulamalardan yararlanarak, arka plandaki yorumlayıcılara da erişebilirler.

ZAYIFLIK

Eğer kullanıcı, onaylanmış veya kodlanmış verinin dışındaki bir veriyi yorumlayıcıya gönderiyorsa, uygulama saldırıya açıktır. Örnek için, aşağıdaki kullanıcı kaynaklı dinamik sorguları kontrol ediniz:

```
PHP:
$sql = "SELECT * FROM table WHERE id = '" . $_REQUEST['id'] . "'";

Java:

String query = "SELECT user_id FROM user_data WHERE user_name = '" +
req.getParameter("userID") + "' and user password = '" + req.getParameter("pwd") +"'";
```

GÜVENLİĞİN DOĞRULANMASI

Burada amaç, kullanıcı verisi değiştirilmeden, uygulama aracılığıyla yorumlayıcıya komut veya sorgu gönderilip gönderilmediğini kontrol etmektir.

Otomatik Yapılan Yaklaşımlar: Çoğu zayıflık tarama aracı enjeksiyon problemlerini tarar, Özellikle de SQL Enjeksiyonunu. Statik analiz araçları, güvenli olmayan yorumlayıcı API'lerini taramak için oldukça kullanışlıdır, fakat sıklıkla doğrulanmış veya kodlanmış koruma yöntemlerine karşı zayıflıkları yakalayamazlar. Eğer uygulama 501 veya 500 dahili sunucu hatası veya detaylı veritabanı hataları veriyorsa, otomatik araçlar ile önlem alınabilir, fakat kod hala risk altında olabilir. Otomatik araçlar LDAP/XML enjeksiyonları ile /XPAth Enjeksiyonlarını saptayabilir.

Elle Yapılan Yaklaşımlar: Bu yorumlayıcıya gönderilen kodlar ile çok etkili ve tam bir kontrol sağlar. Bu işlemin eleştirilecek yanı ise Güvenli API'lerin kullanımının, kodlanmış ve doğrulanmış bölgeler için kontrol edilmelidir. Bu test işlemi son derece fazla zaman alacaktır ve düşük kapsamlı olacaktır, çünkü uygulamaların büyüklüğü nedeniyle saldırı yüzeyi çok geniş olacaktır.



KORUNMA

Yorumlayıcıların kullanımından kaçınılmalıdır. Eğer yorumlayıcıya istek gönderecekseniz, enjeksiyondan kaçınmanın anahtar yöntemi API'lerin kullanımıdır, Örneğin güçlü sorgu ve ilişkili nesne haritalama kütüphaneleri (ORM) kullanılabilir.

Bu arayüzler, bütün verileri gözden geçirir veya bu geçişleri gerektirmeyebilir. Önemli olan arayüzlerin güvenliğine karşın, problem saldırıları saptamak için belirli bir metodun olmamasıdır.

Yorumlayıcıların kullanımı tehlikelidir, öyle ki bu kullanımın önemi, ek dikkat gerektirir:

- Girilen verilerin doğrulanması: Bunun için, öncelikle gelen verilerin uzunluğu, tipi, sözdizimi 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 hata mesajlarının içeriği saldırganın işine yarayacak bilgi içermemelidir.
- API'lerde güçlü sorgu parametre tipleri kullanılmalıdır. Depolanmış prosedürler çağrıldığında, işaretçilerin yerine güçlü tipteki sorgular koyulmalıdır.
- 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.
- Kayıt edilmiş prosedürler kullanılmalıdır. Çünkü genellikle SQL enjeksiyonda bu yöntem güvenlidir. Bunun yanında, dikkatli bir şekilde enjeksiyon yapılabilir (Örneğin exec() kullanılarak veya kayıtlı prosedürleri içeren argümanlar birbirine bağlanması ile...)
- 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.
- Basit escape mekanizması kullanıldığı zaman , basit escape fonksiyonları, escape tablo isimleri olmamalıdır. Tablo isimleri SQL'e uygun isimlerde olmalı ve bu kullanıcı kaynaklı veri girişleri ile ilgili olmamalıdır.
- Varsayılan hata çıktılarını izlenmesi: Çoğunlukla sisteme gelen verilerin kodu çözümlenmeli ve ardından kullanılan dahili yazılım ile gösterilmeden önce kontrol edilmelidir. Elbette ki uygulamanız iki kez girilen benzer veri girişlerinin kodu çözümlenemeyebilecektir. Bu şekilde tehlikeli veri girişleri ile hatalar kullanılarak, kontrol işlemini gerçekleştiren beyaz liste şemaları atlatılabilir.

Kullanılan dil için belirli öneriler:

Java EE- Güçlü tipte PreparedStatement kullanılmalı veya ORM'lerdeki Hibernate veya spring...

- NET- Güçlü tipte sorgular kullanılmalıdır, örneğin SqlCommand ile SqlParameter veya ORM benzeri hibernate'ler.
- PHP- PDO ile güçlü tipteki sorgular kullanılmalı (bindParam() kullanılabilir)

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5121
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4953
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4592

REFERANSLAR

- CWE: CWE-89 (SQL Enjeksiyonu), CWE-77 (Komut Enjeksiyonu), CWE-90 (LDAP Enjeksiyonu), CWE-91 (XML Enjeksiyonu), CWE-93 (CRLF Enjeksiyonu), diğerleri.
- WASC Tehlike sınıflandırması:

http://www.webappsec.org/projects/threat/classes/ldap_injection.shtml

http://www.webappsec.org/projects/threat/classes/sql_injection.shtml

- OWASP, http://www.owasp.org/index.php/SQL_Injection
- OWASP Rehberi, http://www.owasp.org/index.php/Guide_to_SQL_Injection
- OWASP Kod gözden geçirme rehberi , http://www.owasp.org/index.php/Reviewing Code for SQL Injection
- OWASP Test rehberi, http://www.owasp.org/index.php/Testing for SQL Injection
- SQL Enjeksiyon, http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf
- Gelişmiş SQL Enjeksiyon, http://www.ngssoftware.com/papers/advanced-sql-injection.pdf
- Çok gelişmiş SQL Enjeksiyonu, <u>http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf</u>
- Hibernate, gelişmiş bir nesne ilişkilendirme yönetimi (ORM) için J2EE ve .NET, http://www.hibernate.org/
- J2EE Hazırlanmış anlatımlar, http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html
- ASP.Net'de SQL Enjeksiyonlardan nasıl korunurum,

http://msdn2.microsoft.com/en-us/library/ms998271.aspx

PHP PDO fonksiyonları, http://php.net/pdo



A3- ZARARLI DOSYA ÇALIŞTIRMA

Zararlı dosya çalıştırma zayıflığı, çok sayıda uygulamada bulunur. Bu açık, geliştiricilerin çoğunlukla sisteme girilen dosyalara güvenmelerinden kaynaklı olduğundan, potansiyel olarak zararlı olabilecek girişler ile buna bağlı fonksiyonları da beraberinde kabul etmelerinden kaynaklanır. Birçok platformda kullanılan uygulamalarda, URL'ler veya sistem kaynakları gibi, dış kaynaklı nesnelerin kullanımına izin verilir. Veriler yeterince incelenmediğinde, izinsiz erişimlere ve kötü niyetli içerik ile web sunucuya istek göndermeye neden olur.

Bu 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.

Bu saldırı özellike PHP'de yaygındır, ve üst düzeyde dikkat gerektirir. Bu açık tipi, stream veya dosya fonksiyonu kullanılması ile önem kazanmaktadır. Burada kullanıcı kaynaklı dosya isimleri kullanıcının inisiyatifine bırakılmaması gerekir.

ETKİLENEN ORTAMLAR

Kullanıcıdan dosya alan veya dosya isimlendirmeye izin veren tüm web uygulama platformları uzaktan dosya çalıştırma açığına sahip olabilir. Tipik örnek içeriği : .NET platformunda, URL dosya isimlerinin özellikleri ve yerel olarak seçtiği dosya ile aynı isimde olup olmamasını belirlemek kullanıcının isteğine bağlıdır.

PHP özellikle saldırıya açıktır: PHP, uzaktan dosya içerik (RFI) saldırıları ile herhangi bir dosya veya akış (stream) tabanlı API ile sisteme zarar verilebilir.

ZAYIFLIK

Ortak zayıflık yapısı:

```
include $_REQUEST['filename'];
```

Bu sadece uzaktaki saldırgan betiklerin değerlendirilmesini sağlamaz,aynı zamanda yerel dosya sunucularına erişmesini (Eğer PHP Windows işletim sistemi üzerinde çalışıyorsa) PHP'nin dosya sistemindeki SMB desteğinden dolayı sağlar.

Saldırı içeren diğer yöntemler:

- 1. Kötü niyetli oturum dosyaları, günlük verileri veya resimler sisteme yüklenen platformlar (Tipik forum yazılımları)
- 2. Sıkıştırma veya gerçek zamanlı ses yayını yapmakta kullanılan, zlib://veya ogg://gibi PHP URL bayrağının incelenmediği ve uzaktan kaynak kullanımına (_url_fopen veya allow_url_include) izin verildiği platformlar.
- 3. php://input veya POST isteğinden veri alınan PHP wrapper'ları kullanıldığında

4. PHP verisi kullanıldığında: wrapper, data:;base64,PD9waHAgcGhwaW5mbygpOz8+ gibi...

Bu liste daha da uzatılabilir. (ve periyodik olarak güncellenebilir). Burada önemli olan, güvenliği sağlam olarak tasarlanmış bir yapıda, kullanıcı kaynaklı veri girişlerinin, sunucu tarafındaki dosya isimleri ve erişimleri ile etkileşimini göz önünde bulundurmaktır.

PHP örneklerine karşın bu saldırılar farklı yollarla .NET ve J2EE platformlarına da yapılabilir. Bu platformlarda uygulamalar yazılırken özellikle kod erişim güvenliği mekanizmasına dikkat edilmeli,dosya isimleri ihtiyacı karşılayacak biçimde yapılandırılmalı ve kullanıcıya güvenlik kontrollerine etki edebileceği izinler verilmemelidir.

Örneğin, bir saldırgan XML ayrıştırıcısını kullanarak kötü niyetli bir DTD nesnesini yükleyebilir. Bir Avustralyalı güvenlik firması bu yöntemi kullanarak, güvenlik duvarının ardındaki ağ kapılarının (port) nasıl tarandığını göstermiştir. Bunun için bölüm referanslarına [SIF01] bakınız.

Bu zayıflıktan kaynaklı oluşacak hasar, kullanıcının erişim bölgeleri ile sistemin arasındaki izolasyonun kontrolü ile direk ilişkilidir. PHP çok düşük bir izolasyona sahiptir ve kullanıcı erişim bölgesi yapısında veya güvenlik mimarisinde böyle bir izolasyon yoktur.

Zararlı olabilecek saldırılar diğer platformlarda sınırlı veya kısmen güvenlidir. Kapsamlı bir kullanıcı erişim bölgesi bir JVM platformunda, güvenlik yöneticisi çalışır durumda ve doğru bir şekilde yapılandırılmış ise mümkündür (ki buda nadiren olur).

GÜVENLİĞİN DOĞRULANMASI

Otomatik yapılan yaklaşımlar: Zayıflık tarama araçları, bir dosya içeriği veya sistemde çalışması için düzenlenmiş söz dizimi kullanmak suretiyle güçlü bir tanımlama yeteneğine sahip olacaktır. İstatistik analiz araçları API'lerin tehlikeli kullanımını tarayabilir, fakat güvenli bir alanda olduğu düşünülmesine karşın, onaylanmış, kodlanmış alanlardaki açıkları saptayamaz.

Elle yapılan yaklaşımlar: Zayıflığın bulunması için uygulama içerisine izin verilmiş özel kod taşıyan bir dosya dahil edilebilir, fakat bu bilinen hataların bulunmasını sağlayacaktır. Bu şekilde zayıflıklar test edilebilir, fakat zayıflığın tanımlanması, özel parametreler ve doğru söz dizimi gerektirdiğinden zor bir yöntemdir.

KORUNMA

Uzaktan çalışabilir içeriğin önlenmesi için uygulama mimarisinin ve tasarım evrelerinin dikkatle planlanması ve tam bir testten geçmesi gerekmektedir. Genellikle, iyi yazılmış uygulamalar kullanıcı kaynaklı veri girişlerinde dosya ismi ve herhangi sunucu tabanlı kaynakların (örneğin resimler, XML ve XLS dönüştürülmüş dokümanlar veya betik eklentileri) kullanımına izin vermemenin yanında, güvenlik duvarına, İnternet veya yerel ağ ile diğer sunuculara yeni bağlantılar oluşturulmasını engelleyen kurallar eklenmelidir. Bununla beraber, kullanıcı kaynaklı girişlerde ihtiyaç duyulan uygulamalar çalışmaya devam edecektir.

Dikkat edilmesi gereken noktalar:

• Bir dolaylı nesne referans haritası kullanın (Indirect Object Reference Map)

(Daha fazla bilgi için A4 bölümüne bakınız). Örneğin, bir dosya ismi bir kez kullanılsın ve referans olarak kullanılacak bir bölüm HASH değerini sağlasın.



```
<select name="language">
<option value="English">English</option>

Yukarıdaki "tag"yerine

<select name="language">
<option value="78463a384a5aa4fad5fa73e2f506ecfc">English</option>
```

kullanımı gibi.

Dolaylı nesne referanslarının birer birer deneme (brute force) saldırısı ile zayıflayacağı dikkate alınmalıdır. Buna alternatif olarak, 1,2,3 gibi bir indeks değeri kullanan ve sıralı bir şekilde devam eden parametreler kontrol edilebilir.

 Belirgin bir kusur kontrol mekanizması kullanın: Kullandığınız dil destekliyorsa, kusur kontrol mekanizması kullanın. Başka bir şekilde bir değişken isimlendirme şeması yardımıyla kusurları kontrol edin:

```
$hostile = &$_POST; // refer to POST variables, not $_REQUEST
$safe['filename'] = validate_file_name($hostile['unsafe_filename']); //
make it safe
```

Bu yüzden, kötü niyetli operasyonlar açıkça tanımlanmalı:

```
require_once($_POST['unsafe_filename'] . 'inc.php'); (Yanlış)
require_once($safe['filename'] . 'inc.php'); (Doğru)
```

- Kullanıcı girişlerini kontrol edecek güçlü mekanizmalar kullanın ve gelen verinin "iyi olduğu kabul edildi" stratejisi oluşturun.
- Firewall'a kurallar ekleyin: Harici web sitelerine ve dahili sistemlere yeni bağlantıları engelleyecek firewall kuralları oluşturun. Yüksek güvenlikli sistemler için, VLAN veya özel alt ağ (subnet) ile web sunucusunu izole edin.
- Kullanıcı kaynaklı dosyaları ve dosya isimlerini kontrol edin: Kontrol etmenin yanında diğer kontrolleri devre dışı bırakın, örneğin oturum nesnesindeki veriler bir kusurdur. Bunun yanında değişkenler, resimler, PDF raporları, geçici dosyalar, vb. içerikte kusur içerecektir.
- Bir chroot veya diğer kullanıcı alanı tanımlama mekanizmaları kullanılmalı. Diğer uygulamalardan kullanıcıları izole edebilecek sanal makine yapısı kullanılabilir.
- PHP: allow_url_fopen ve allow_url_include değerlerini, php.ini dosyasından kapalı duruma getirin ve PHP'nin bu fonksiyonu yerel olarak içermediğini de kontrol edin. Birkaç uygulama için, bu işlev ve ayarlar açık olması gerekebilir.
- PHP: register_globals değerini kapatın, E_STRICT değerini kullanarak başlangıç değerinde olmayan değişkenleri bulun.

 PHP: Bütün dosyaları ve stream fonksiyonlarını (stream_*) dikkat ile inceleyin. Bu fonksiyonlar kullanıcıların gönderdiği veriler içerisinde herhangi bir dosya ismi argümanı almamasını sağlamak amacıyla kontrol edilmelidir:

```
include() include_once() require() require_once() fopen()
imagecreatefromXXX() file() file_get_contents() copy() delete()
unlink() upload tmp dir() $ FILES move uploaded file()
```

Kullanılan dil için belirli öneriler:

- PHP: system() eval() passthru() veya backtick operatörü içeren veriniz varsa çok dikkatli olmanız gerekmektedir.
- J2EE için "güvenlik yönetimi" çalışır durumda olmalı ve düzgün bir biçimde yapılandırılmalı, uygulamanın izinleri gözden geçirilmelidir.
- ASP.NET için lütfen referans dokümanlarına bakın. Uygulamanızı tasarlarken parçalara ayırarak güvenliğini sağlayın, böylece uygulamanızın büyük bölümü içerisindeki olası zayıflıkları izole edebilirsiniz.

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0360
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5220
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4722

REFERANSLAR

- CWE:CWE-98 (PHP dosya alma), CWE-78(OS Komut enjeksiyonu), CWE-95(Eval Enjeksiyonu),
 CWE-434 (Sınırsız dosya yükleme)
- WASC Tehlike sınıflandırması:

http://www.webappsec.org/projects/threat/classes/os_commanding.shtml

- OWASP Rehberi, http://www.owasp.org/index.php/File_System#Includes_and_Remote_files
- OWASP Test rehberi, http://www.owasp.org/index.php/Testing for Directory Traversal
- OWASP PHP Top 5, http://www.owasp.org/index.php/PHP_Top_5#P1: Remote_Code_Execution
- Stefan Esser,

http://blog.php-security.org/archives/45-PHP-5.2.0-and-allow_url_include.html

• [SIF01] SIFT, Web Servisleri: Kuralların değişimi,

http://www.ruxcon.org.au/files/2006/web_services_security.ppt



- http://www.owasp.org/index.php/OWASP_Java Table of Contents#Defining a Java Security Policy
- Microsoft Kısmen etkili güvenlik için programlama,

http://msdn2.microsoft.com/en-us/library/ms364059(VS.80).aspx

A4- EMNİYETSİZ DOĞRUDAN NESNE REFERANSI

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.

Saldırının bu tipi, Avustralya vergilendirme ofisi GST'nin başlangıç yardımcı sitesinin başına 2000 yılında geldi; kayıtlı fakat kötü niyetli olan bir kullanıcı, URL içinde verilen (şirket vergi tanıtıcısı olan) ABN'yi basitçe değiştirdi. Kullanıcı, sistemden yaklaşık 17,000 şirkete ait tüm bilgilere el koydu ve sonra saldırısının ayrıntılarını 17,000 şirketin her birine e-posta ile gönderdi. Saldırıya açıklığın bu tipi oldukça yaygındır, ama birçok uygulamada büyük ölçüde denenmemiştir.

ETKİLENEN ORTAMLAR

Bütün İnternet uygulama yapıları, emniyetsiz doğrudan nesne referanslarında, saldırılara karşı saldırıya açıktır.

SALDIRIYA AÇIKLIK

Birçok uygulama, kullanıcılara iç nesne referanslarını açıklar. Saldırganlarda referansları değiştirmek için parametrelerle oynarlar ve tasarlanmış fakat güçlendirilmemiş erişim kontrol politikasını bozarlar. Genellikle bu referanslar dosya sistemlerini ve veritabanlarını işaret eder, fakat açığa vurulan herhangi bir uygulama yapısının saldırıya açık olma ihtimali de vardır.

Örneğin, eğer kod, dosya adları veya yolları belirtmesi için kullanıcı girişine izin veriyorsa, uygulamanın çalışma dizininden dışarı ulaşması ve diğer kaynaklara erişmesi için saldırganlara da izin verebilir.

```
<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 (Daha çok bilgi için OWASP rehberine bakın) erişmek için "../../../etc/passwd%00" gibi bir dizgi kullanarak, <u>null byte injection</u> yöntemiyle (boş byte saldırısı) saldırılabilir.

Benzer şekilde, veritabanı anahtarlarına olan referanslar da sık sık açığa çıkar. Bir saldırgan, basitçe tahmin ederek veya başka bir geçerli anahtar arayarak bu parametrelere saldırabilir. Çoğunlukla bunlar doğada sıralıdır. Aşağıdaki örnekte, uygulama, yetkisiz kartlar için herhangi bir bağlantıyı sunmasa ve hiçbir SQL saldırısı mümkün olmasa da saldırgan cartID parametresini değiştirerek istediği herhangi bir kartı elde edebilir.

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
```



```
String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

GÜVENLİĞİ DOĞRULAMAK

Amaç, uygulamanın, bir saldırgan tarafından yönetilmesi için doğrudan nesne referanslarına izin vermediğini doğrulamaktır.

Otomatik yapılan yaklaşımlar: Saldırıya açıklık tarama araçları, hangi parametrelerin hile karıştırılmaya yatkın olduğunu veya hile karıştırmanın çalışıp çalışmadığını anlamakta sorun yaşayacaktır. Saldırıya açıklık tarama araçları, parametrelerin, yönetilmeye hassas olduğu veya değiştirilerek çalışıp çalışmadığını anlayacak zorlu tanımalara sahip olmalıdır. Statik analiz araçları kullanımdan önce hangi parametrelerin erişim kontrol denetimine sahip olması gerektiğini bilemez.

Elle yapılan yaklaşımlar: Kod incelemesi, kritik parametreleri izleyebilir, birçok olayda yönetilmeye hassas olup olmadığı teşhis edebilir. İçeri sızma testi, yönetilebilirliğin mümkün olup olmadığını da doğrulayabilir. Yine de, bu tekniklerin her ikisinin çalışması da zaman alabilir ve yetersiz olabilir.

KORUMA

En iyi koruma, indeksleme, dolaylı referans haritası veya onaylanması kolay diğer dolaylı metotlar kullanılarak kullanıcılara doğrudan nesne referanslarını açığa vurmaktan kaçınmaktır. Eğer doğrudan nesne referansı kullanılması şart ise, kullanıcının referansı kullanmadan önce kimlik doğrulamadan geçtiğinden emin olunmalıdır.

Uygulama nesnelerine başvurmanın standart bir yolunu kurmak önemlidir:

- Kullanıcılara mümkünse özel nesne referanslarını açıklamadan kaçının, örneğin birincil anahtarlar veya dosya adları gibi;
- Herhangi bir özel nesne referansını "iyi bilineni kabul et" yaklaşımı ile yaygın olarak geçerli kıl;
- Bütün referans gösterilen nesnelerde kimlik doğrulamasını doğrula.

Parametre yönetimi saldırılarını engellemek için en iyi çözüm, indeks değeri veya referans haritası kullanmaktır.

```
http://www.example.com/application?file=1
```

Eğer veritabanı yapılarına olan doğrudan referanslar açığa vurulmak zorundaysa, SQL ifadelerinde ve diğer veritabanı erişim metotlarında sadece kimlik doğrulama yapılmış kayıtlara izin verildiğinden emin olunmalı:

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
User user = (User)request.getSession().getAttribute( "user" );
String query = "SELECT * FROM table WHERE cartID=" + cartID + " AND userID=" + user.getID();
```

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0329
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4369

• http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0229

REFERANSLAR

- CWE: CWE-22 (Path Traversal), CWE-472 (internet Parameter Tampering)
- WASC Threat Classification:
 http://www.internetappsec.org/projects/threat/classes/abuse_of_functionality.shtml
 http://www.internetappsec.org/projects/threat/classes/insufficient_authorization.shtml
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing for business logic
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing for Directory Traversal
- OWASP, http://www.owasp.org/index.php/Category:Access Control Vulnerability
- GST Assist attack details, http://www.abc.net.au/7.30/stories/s146760.htm



A5- SİTELER ÖTESİ İSTEK SAHTECİLİĞİ (CSRF)

Siteler ötesi istek sahteciliği (Cross Site Request Forgery) yeni bir saldırı türü değildir fakat basittir ve zarar vericidir. CSRF saldırısı, saldırıya açık bir İnternet uygulamasına bir isteği yollaması için kurbanın tarayıcısına zorla giriş yapar, sonra kurbanın adına seçilen hareketi yapar.

Bu saldırıya açıklık aşırı derecede yaygındır, herhangi bir İnternet uygulaması şu nedenlerden dolayı risk altındadır;

- Saldırıya açık işlemler için hiçbir kimlik doğrulama kontrolü yoktur,
- Eğer orijinal giriş bilgileri (kullanıcı adı & şifre) geçerliyse, işlem gerçekleşecektir, (Örneğin http://www.example.com/admin/doSomething.ctl?username=admin&passwd=admin)
- Yetkilendirilmiş istekler, eğer oturum açılmış uygulama varsa oturum çerez bilgileri, eğer oturum açılmamışsa "Beni hatırla" fonksiyonu, veya eğer aktif dizinle açılmış oturumda katılımcı bir Intranet'in parçası ile bütünleşmişse Kerberos işareti gibi otomatik olarak yollanan kimlik bilgilerine dayanır.

Maalesef bugün birçok İnternet uygulaması sadece, otomatik olarak yollanan, oturum çerezleri, temel kimlik doğrulama bilgileri, kaynak IP adresleri, SSL sertifikaları veya Windows alan güven belgelerine güvenmektedir.

Bu saldırıya açıklık ayrıca "Session Riding, One-Click Attacks, Cross Site Reference Forgery, Hostile Linking, and Automation Attack" gibi birkaç başka isim ile de anılır. Baş harflerinin kısaltması olan XSRF de sıklıkla kullanılır. OWASP ve MITRE, Cross Site Request Forgery (siteler ötesi istek sahteciliği) ve CSRF terimlerinin her ikisini de standartlaştırmıştır.

ETKİLENEN ORTAMLAR

Bütün İnternet uygulama yapıları, CSRF'e karşı saldırıya açıktır.

SALDIRIYA AÇIKLIK

Tipik olarak bir foruma karşı yapılan CSRF saldırısında bazı fonksiyonları çağırmak için, oturum kapatma sayfasında olduğu gibi, yönlendiren kullanıcı formunu alabilir. Kurban tarafından görülen aşağıdaki etikete sahip herhangi bir İnternet sayfasında 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">
```

"BlackHat-2006" da "Intranet sitelerine dışarıdan saldırmak" konusunda konuşan Jeremiah Grossman, kullanıcı, DSL yönlendiricisinin bir İnternet arayüzü olduğunu bilmezse dahi, rızası olmadan DSL yönlendiricisinde istediği değişiklikleri yapmak için bir kullanıcıyı zorlamanın mümkün olduğunu gösterdi. Jeremiah saldırıyı gerçekleştirmek için yönlendiricinin hazır gelen hesap ismini kullandı.

Bu saldırıların hepsi işe yarar, çünkü saldırgan izin belgesini sağlamasa da, kullanıcının izin belgesi otomatik olarak tarayıcının isteğine dahil olur (genellikle oturum çerezi).

Eğer saldırıyı içeren etiket, saldırıya açık uygulamaya gönderilebilirse, o zaman kurbanlarda kaydedileni bulmanın olasılığı, depolanan ve yansıtılan XSS (Cross Site Scripting - siteler arası betik yazma) kusurları arasındaki riskteki artışa benzer şekilde artar. XSS kusurları, XSS kusurlu uygulamanın CSRF'e hassas olmasına rağmen, CSRF saldırısını çalıştırabilmek için gerekli değildir. Çünkü CSRF saldırısı, kendisine karşı koymak için basamak olabilen elle gönderilen herhangi bir izin belgesini çalmak için XSS kusurunu kullanabilir. Birçok uygulama solucanı, her iki tekniğin birleşimini kullanmıştır.

CSRF saldırısına karşı savunma inşa etmek için, uygulamada XSS açıklıklarını ortaya çıkarmaya odaklanmalı, çünkü böyle kusurlar, CSRF savunmalarını aşmak için kullanılabilir.

GÜVENLİĞİ DOĞRULAMAK

Amaç, CSRF saldırılarına karşı uygulamayı, tarayıcı tarafından otomatik olarak yollanmayan kimlik doğrulama işaretlerini oluşturarak ve isteyerek korumaktır.

Otomatikleştirilen yaklaşımlar: CSRF'nin uygulama tarama motorları tarafından tespiti mümkün olmasına rağmen bugün az sayıda otomatik tarayıcı CSRF saldırısı açıklığını tespit edebilmektedir. Ancak eğer uygulama tarayıcısı, siteler arası betik yazma saldırısına açıklığı algılasa da ve CSRF'e karşı koruma yoksa, önceden hazırlanmış paket CSRF saldırılarına karşı risk büyük olasılıkla vardır.

Elle yapılan yaklaşımlar: Yayılma testi, CSRF korumasının yerinde doğrulaması için hızlı bir yoldur. Bu mekanizmanın kuvvetli ve uygun şekilde olduğunu doğrulamanın en verimli yöntemi kodu kontrol etmektir.

KORUMA

Uygulamalar, tarayıcılar tarafından otomatik olarak yollanan izin belgesi veya işaretlerine güvenmemelidir. Uygulamanın CSRF saldırısına karşı koruma içermesinin tek çözümü, tarayıcının hatırlayamayacağı özel bir işareti kullanmaktır.

Takip eden stratejiler, bütün İnternet uygulamalarında zaten olmalıdır:

- Uygulamada hiçbir XSS saldırıya açıklığı olmamalıdır (Bakınız A1 Siteler Arası Betik Yazma)
- 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>
```

Ve sonra güncel kullanıcı için gönderilen işaretin doğru olduğunu kanıtlanmalıdır. Bunun gibi işaretler kullanıcı için bazı özel sayfa veya fonksiyonlarda ya da basitçe bütün oturumda benzersiz olabilir. İşaretlerin özel bir fonksiyon ve/veya özel veri takımı olmasına odaklanmak daha kuvvetli bir koruma sağlasa da yapıyı insa etmek ve sürdürmek daha zor olacaktır.



- Hassas veri veya değer işleri için, tekrar kimlik doğrulanmalı, veya isteğin gerçek olduğunu garanti etmesi için işlem işareti kullanılmalıdır. İstekleri doğrulamak veya kullanıcı isteğini iletmek amacıyla e-posta veya telefon irtibatı gibi dış mekanizmalar kurulmalıdır.
- 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.
- POST metodu tek başına yetersiz bir korumadır. CSRF'e karşı uygun şekilde korunmak için bant dışı kimlik doğrulamasıyla, tekrar kimlik doğrulamayla veya rastgele işaretlerle birleştirilmelidir.
- ASP.NET için, ViewStateUserKey kurulmalı (Bakınız Referanslar). Bu işlem, yukarıda tanımladığını gibi rastgele bir işareti benzer yöntemle kontrol etmeyi sağlar.

Bu öneriler, saldırılara maruz kalmayı önemli ölçüde azaltacaksa da, gelişmiş CSRF saldırıları, bu kısıtlamaların birçoğunu geçebilir. En kuvvetli teknik, benzersiz işaretleri kullanmak ve uygulamadaki bütün XSS saldırıya açıklıklarını yok etmektir.

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0192
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5116
- MySpace Samy Interview: http://blog.outer-court.com/archive/2005-10-14-n81.html
- An attack which uses Quicktime to perform CSRF attacks
 http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9005607&intsrc=hm-list

REFERANSLAR

- CWE: CWE-352 (Cross-Site Request Forgery)
- WASC Threat Classification: No direct mapping, but the following is a close match: http://www.internetappsec.org/projects/threat/classes/abuse of functionality.shtml
- OWASP CSRF, http://www.owasp.org/index.php/Cross-Site Request Forgery
- OWASP Testing Guide, https://www.owasp.org/index.php/Testing for CSRF
- OWASP CSRF Guard, http://www.owasp.org/index.php/CSRF Guard
- OWASP PHP CSRF Guard, http://www.owasp.org/index.php/PHP CSRF Guard
- RSnake, "What is CSRF?", http://ha.ckers.org/blog/20061030/what-is-csrf/
- Jeremiah Grossman, slides and demos of "Hacking Intranet sites from the outside" http://www.whitehatsec.com/presentations/whitehat bh pres 08032006.tar.gz
- Microsoft, ViewStateUserKey details, http://msdn2.microsoft.com/en-us/library/ms972969.aspx#securitybarriers topic2

A6- BİLGİ SIZINTISI VE UYGUNSUZ HATA IŞLEME

Uygulamalar, yapılandırmaları ve iç çalışmaları ile ilgili bilgiyi istemeyerek sızdırabilir, veya çeşitli uygulama problemleriyle gizliliği bozabilir. Uygulamalar ayrıca, belli işlemleri yürütmenin ne kadar sürdüğüyle, ya da farklı girişlere farklı çıkışlar vermekle, örneğin farklı hata numaralarıyla aynı hata metnini göstermek gibi durumlarda, iç durumu sızdırabilir. İnternet uygulamaları çoğunlukla ayrıntılı hata mesajları veya hata ayıklama modundaki hata mesajları vasıtasıyla iç durumları hakkında bilgi sızdırırlar. Çoğunlukla bu bilgi daha güçlü saldırıları gerçekleştirmekte veya hatta saldırıları otomatikleştirmekte kullanılabilir.

ETKİLENEN ORTAMLAR

Bütün İnternet uygulama yapıları, bilgi sızmasına ve uygunsuz hatayı işlemeye karşı saldırıya açıktır.

SALDIRIYA AÇIKLIK

Uygulamalar sık sık hata mesajı üretir ve bunları kullanıcılara gösterir. Birçok defa bu hata mesajları, saldırganlar için çok faydalıdır, çünkü bu hata mesajları saldırıya açıklığı sömürmek için faydalı olan bilgiyi veya uygulama ayrıntılarını gösterir. Bunun birkaç yaygın örneği vardır:

- Hatanın çok fazla bilgiyle, örneğin yığın izleme, başarısız SQL ifadeleri veya diğer hata ayıklama bilgileriyle, gösterilmesini sağlayan <u>Detaylı Hata İşleme</u> (Detailed Error Handling)
- Farklı girişlere farklı sonuçlar üreten fonksiyonlar. Giriş fonksiyonuna sağlanan aynı kullanıcı ismi ve farklı şifrelerde, var olmayan kullanıcı veya hatalı şifreler için aynı metni üretmelidir. Ancak birçok sistem, farklı hata kodları üretir.

GÜVENLİĞİ DOĞRULAMAK

Amaç, uygulamanın, hata mesajlarıyla veya diğer yollarla bilgiyi sızdırmadığını doğrulamaktır.

Otomatik yapılan yaklaşımlar: Saldırıya açıklık tarama araçları genellikle hata mesajları oluşturulmasına sebep olur. Statik analiz araçları, bilgiyi sızdıran API'nin (uygulama programlama arabirimi) kullanımı için arama yapabilir fakat o mesajların anlamını doğrulayamayacaktır.

Elle yapılan yaklaşımlar: Bir kod incelemesi, bilgiyi sızıntısı ve uygunsuz hata işlenmesi ve diğer örnekler için araştırma yapılabilir fakat bu çok zaman alacaktır. Ayrıca test ederek de hata mesajları oluşturulabilir, fakat hangi hata yollarının kapsandığını bilmek oldukça zordur.

KORUMA

Geliştiriciler OWASP'ın WebScarab gibi araçlarını uygulamalarına hata ürettirmek için kullanmalı. Bu yolla test edilmeyen uygulamalar kesinlikle, beklenilmeyen hata çıktıları oluşturacaktır. Ayrıca uygulamalar, saldırganlara istenmeyen bilgiyi sızdırmaması için standart kural dışı durum işleme mimarisini içermelidir.

Bilgi sızmasını engellemek, disiplini gerektirir. Aşağıdaki örnekler, etkili bir şekilde kanıtlanmıştır:

- Tüm yazılım geliştirme takımının kural dışı durum işlemesi için ortak yaklaşımı paylaştığı garanti edilmeli;
- Detaylı hata işleme sınırlandırılmalı veya iptal edilmeli. Özellikle son kullanıcıya ayrıntılanmış bilgi, yığın bilgileri veya yol bilgisi gösterilmemeli



- Yaklaşık aynı zamanda benzer veya aynı hata mesajlarını veren, çeşitli sonuçlara sahip olan güvenlik yolları sağlanmalı. Eğer bu mümkün değilse, saldırgan'dan bu ayrıntıyı saklamak adına bütün kayıtlar için rastgele bir bekleme zamanı konulmalı;
- Çeşitli katmanlar, ölümcül veya olağan dışı sonuçlar verebilir, örneğin veritabanı katmanı, temel İnternet sunucusu (IIS, Apache, gibi). Bütün tabakalardaki hataların, saldırgan tarafından sömürülmesini engellemek için, yeterince kontrol edilmiş ve yapılandırılmış olması çok önemlidir;
- Yaygın yapıların hatanın iskelet kodunun içinde veya genel kodun içinde olmasına göre farklı HTTP hata kodları verdiğinin bilincinde olunmalı. Kullanıcıların büyük kısmı için, tüm hata üretim yollarında, uygun bir şekilde zarar verici bilgilerden arındırılmış bir varsayılan hata işleyici yaratmak oldukça faydalıdır.
- Tanımlı hata işleyiciyi daima "200" (OK) hata ekranını döndürmesi için geçersiz kılmak, otomatik tarama araçlarının ciddi hataları tespit yeteneğini azaltır. Bununla birlikte, bu "gizlilik üzerinden güvenlik" savunmaya fazladan bir savunma katmanı sağlayabilir;
- Bazı büyük organizasyonlar, bütün uygulamaların arasına rastgele/benzersiz hata kodları eklemeyi seçebilir. Bu, özel bir hata için doğru çözümü bulmakta yardım masasına (help desk) yardımcı olabilir, ama uygulamanın hangi yolda başarısız olduğu tam olarak belirleyebilmesi için saldırganlara da fırsat verebilir.

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4899
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3389
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0580

REFERANSLAR

- CWE: CWE-200 (Information Leak), CWE-203 (Discrepancy Information Leak), CWE-215 (Information Leak Through Debug Information), CWE-209 (Error Message Information Leak), others.
- WASC Threat Classification: http://www.internetappsec.org/projects/threat/classes/information_leakage.shtml
- OWASP, http://www.owasp.org/index.php/Error_Handling
- OWASP, http://www.owasp.org/index.php/Category:Sensitive Data Protection Vulnerability

A7- IHLAL EDILMIŞ KİMLİK DOĞRULAMA VE OTURUM YÖNETİMİ

Uygun kimlik doğrulama ve oturum yönetimi, İnternet uygulama güvenliği için kritiktir. Bu alandaki kusurlar sıklıkla, kimlik bilgisi ve oturum işaretlerini kendi döngüleri boyunca korumada bozukluğu ihtiva eder. Bu kusurlar, kullanıcı veya idare hesapları korsanlığına, kimlik doğrulama ve hesap sorumluluğu kontrollerinin çökmesine ve gizlilik ihlallerine sebep olabilir.

ETKİLENEN ORTAMLAR

Bütün İnternet uygulama yapıları, kimlik doğrulama ve oturum idare kusurlarına karşı korunmasızdır.

SALDIRIYA AÇIKLIK

Ana kimlik doğrulama mekanizmasında kusurlar, seyrek değildir, fakat zayıf noktalar daha çok, çıkış, parola yönetimi, zaman aşımı, beni hatırla, gizli soru ve hesap güncelleştirmesi gibi yardımcı kimlik doğrulama fonksiyonları boyunca ortaya çıkar.

GÜVENLİĞİ DOĞRULAMAK

Amaç, uygulamanın, kullanıcıların kimliklerini uygun şekilde doğruladığını ve kimlikleri ve kimlik bilgilerini uygun şekilde koruduğunu doğrulamaktır.

Otomatik yapılan yaklaşımlar: Saldırıya açıklık tarama araçları, özel kimlik doğrulama ve oturum yönetim planlarında çok zor zaman tespiti saldırı açıklarına sahiptir. Statik analiz araçları da, özel kodda oturum yönetim ve kimlik doğrulama problemleri bulmak için muhtemel değildir.

Elle yapılan yaklaşımlar: Kod incelemesi ve test etme, özellikle de bileşimi, uygun şekilde yerine getirilen kimlik doğrulama, oturum yönetimi ve yardımcı fonksiyonları doğrulamakta oldukça etkilidir.

KORUMA

Kimlik doğrulama, güvenli haberleşme ve kimlik bilgileri depolamasına dayanır. İlk olarak, SSL'in uygulamanın tüm kimlik doğrulanmış parçaları için tek seçenek olduğu garanti edilmeli, (bakınız A9 – emniyetsiz iletişim), ve bütün kimlik bilgileri, kriptografik özet veya şifrelenen formda depolanmalıdır, (bakınız A8 – Güvensiz kriptografik depolama).

Kimlik doğrulama kusurlarını engellemek, dikkatli bir plan gerektirir. Göz önüne alınması gereken en önemliler arasındakiler olarak aşağıdakiler sayılabilir:

- Sadece yerleşik oturum yönetim mekanizması kullanılmalı. Herhangi bir koşul altında ikincil oturum denetimcisini yazmamalı veya kullanılmamalı;
- URL ile veya istekle gelen yeni, önceden ayarlanmış veya geçersiz oturum tanıyıcıları kabul edilmemeli. Bu saldırı türü, oturum tespit saldırısı olarak isimlendirilir.
- Kimlik doğrulama veya oturum yönetimi amaçları için özel (custom) çerezlerin kodu sınırlanmalı veya defedilmeli, örneğin "Beni Hatırla" tipi veya kurum bilgi sistemleri için tek giriş (sigle sign on) fonksiyonları gibi. Bu, güçlü, iyice kanıtlanmış SSO (Single Sign On tek imzayla giriş / Web Çoklu Oturum Açma) veya birleşmiş kimlik doğrulama çözümlerini uygulanmaz;
- Uygun dayanıklılığa ve etmene bağlı tek bir kimlik doğrulama mekanizması kullanılmalı. Bu mekanizmanın, kolayca yanıltmaya veya tekrarlanan saldırılara maruz bırakılamadığına emin



olunmalı. Bu mekanizma fazla karmaşık olmamalıdır, sonra bu karmaşıklık esas saldırı şekline dönüşebilir.

- Şifrelenmemiş bir sayfadan oturum açma işleminin başlamasına izin verilmemelidir. Kimlik veya oturum bilgilerini çalmayı, parola balıkçılığı (phishing) ve oturum belirleme saldırısını engellemek için yeni veya güncel oturum jetonu (token) ile şifrelenmiş ikinci bir sayfadan oturum açma işlemi yapılmalıdır.
- Başarılı kimlik doğrulama veya hak seviye değişikliği üzerine yeni bir oturum oluşturma göz önüne alınmalıdır.
- Her sayfada bir oturumu kapatma bağlantısı olduğu garanti edilmelidir. Oturum kapatma işlemi, bütün sunucu tarafı oturum durumunu ve kullanıcı tarafı çerezlerini yok etmelidir. İnsan faktörleri düşünülmeli: Doğrulama için soru sorulmamalı çünkü kullanıcılar, başarılı bir şekilde oturumu kapatmak yerine, pencere veya sekme kapatmayı seçecektir.
- Korunan verinin değerine göre etkin olmayan bir oturumdan otomatik olarak dışarı çıkartan bir zaman aşımı periyotu kullanılmalı. (Kısa periyot her zaman daha iyidir)
- Sadece kuvvetli yardımcı kimlik sorgulama fonksiyonları kullanılmalı (soru ve cevaplar, parola yenilemesi gibi), çünkü bunlarda kullanıcı isimleri, parolaları veya jetonlar gibi kimlik bilgisidir. Açığa çıkarma saldırılarını engellemek için cevaplara tek yönlü kriptografik özeti kullanılmalıdır.
- Herhangi bir oturum tanımlayıcısı veya URL veya günlük kayıtları içindeki geçerli kimlik bilgileri herhangi bir kısmı açıklanmamalıdır (günlük dosyalarına kullanıcının şifresi depolanmamalı veya hiçbir oturum tekrar yazılmamalıdır).
- Kullanıcı, şifresini değiştireceği zaman, eski şifresi kontrol edilmelidir.
- Kimlik doğrulamanın biricik biçimi olarak hileli olabilecek kimlik bilgilerine güvenilmemeli, örneğin IP adresleri veya adres aralığı, DNS veya ters DNS aramaları, başvuru başlığı ve benzerleri.
- Kaydedilen e-posta adreslerine, şifre yenilemesi için mekanizma olarak, gizli bilgileri yollarken dikkatli olunmalı (Referanslarda RSNAKE01'e bakınız). Erişimi yenilemek için sadece-sınırlızaman (limited-time-only) rastgele sayıları kullanılmalı ve şifre yenilenir yenilenmez takip eden (bilgilendirici) e-posta gönderilmelidir. E-posta adresini değiştiren kullanıcılar kendi kaydettikleri epostayı kullanırken dikkatli olunmalı, değişikliği uygulamadan önce, önceki e-posta adresine bir mesaj yollanmalıdır.

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6229
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6528

REFERANSLAR

- CWE: CWE-287 (Authentication Issues), CWE-522 (Insufficiently Protected Credentials),
- CWE-311 (Reflection attack in an authentication protocol), others.
- WASC Threat Classification:

http://www.internetappsec.org/projects/threat/classes/insufficient_authentication.shtml http://www.internetappsec.org/projects/threat/classes/credential_session_prediction.shtml http://www.internetappsec.org/projects/threat/classes/session_fixation.shtml

- OWASP Guide, http://www.owasp.org/index.php/Guide to Authentication
- OWASP Code Review Guide, http://www.owasp.org/index.php/Reviewing_Code_for_Authentication
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing for authentication
- RSNAKE01 http://ha.ckers.org/blog/20070122/ip-trust-relationships-xss-and-you



A8- GÜVENSİZ KRİPTOGRAFİK DEPOLAMA

Hassasiyet derecesi yüksek olan verileri kriptografi ile korumak, birçok web uygulaması için anahtar konuma gelmiştir. Basitçe hassasiyet derecesi yüksek verileri şifreleyememek oldukça yaygın bir konudur. İçeriğini sık sık kötü bir şekilde şifreleyen uygulamalar da hem uygun olmayan bir şekilde şifreleme kullanılmakta hem de güçlü şifreleme hataları yapılmaktadır. Bu kusurlar, hassas verinin ortaya çıkmasına ve uyumsuzluk sorununa yol açabilir.

ETKİLENEN ORTAMLAR:

Bütün web uygulamalarının mimarisi, güvensiz şifrelenmiş veri depolanmasına karşı savunmasızdır.

ZAYIFLIK

Şifreleme sorunlarını önlemek için dikkatli bir planlama yapmak gerekmektedir. Yaygınca görülen sorunları şöyle sıralayabiliriz;

- Hassas verileri şifrelememek
- Basit algoritmalar kullanmak
- Güçlü algoritmaları güvensiz bir şekilde kullanmak
- Zayıflığı ispatlanmış algoritmaları kullanmaya devam etmek (MD5, SHA-1, RC3, RC4, etc...)
- Güçlü kodlanmış şifrelerin ve bunların korunmasız olarak depolanması.

GÜVENLİK DOĞRULAMA:

Amaç depolanan düzgünce şifrelenmiş hassas uygulama verisinin onaylanmasıdır.

Otomatik Yapılan Yaklaşımlar: Zayıflık tarama araçları şifrelenmiş veri depolarını asla doğruluğunu kontrol edemeyecektir. Kod tarama araçları bilinen şifrelenmiş API'lerin kullanımını bulabilir ancak veri depolama biriminden farklı bir yerde şifrelenmiş olarak tutuluyorsa düzgün olarak bulunamayabilir.

Elle Yapılan Yaklaşımlar: Sunucular sınanırken şifrelenmiş depo birimi onaylanamayabilir (taramalar gibi). Kodu gözden geçirmek, şifre yönetim mekanizmasının düzgünce gerçekleştirilmesi için hassas veriyi şifreleme uygulamaları en iyi yoludur. Bu, bazı durumlarda dış sistemlerin yapılandırmasının kontrol edilmesine yol açar.

KORUNMA:

En önemli görüş, gerçekte şifrelenmiş her şey emniyete alınmıştır. Ve siz kriptografinin düzgünce yapıldığından emin olmak zorundasınız. Kriptografinin düzgünce kullanılmadığı bir çok yol vardır. Aşağıdaki öneriler kriptografik materyallerin güvenli şifrelenmesine yardımcı olmak için sizin tarama sisteminiz bir parçası olabilir:

- Kriptografik algoritma oluşturmayın: Halka açık (Public) algoritmaları kullanmak (AES, RSA, SHA-256 yada daha iyi bir hash algoritması).
- Zayıf algoritmalar kullanmayın. MD5/SHA-1 gibi zayıf algoritmalar kullanmayın. Daha güvenli alternatifleri olarak SHA-256 yada daha iyisi kullanın.
- Anahtarlarınızı offline üretin ve saklarken ekstra özen gösterin. Güvenli olmayan kanallarda asla kişisel anahtarlarınızı yayınlamayın.
- MQ kuyruk erişim detayları yada veritabanı bilgileri gibi yapı bilgileri garantilenerek düzgün bir şekilde (sıkı dosya izin ve kontrolleri ile) güven altına alınmalıdır veya güvenli bir şekilde şifrelenmiş ve şifresi lokal veya uzak kullanıcılar tarafından açılamamalıdır.
- Disk üzerinde depolanmış şifreli verinin şifresinin çözümlenmesi kolay değildir. Örnek olarak, eğer veritabanı bağlantı havuzu şifrelenmemiş erişime açıksa veritabanı şifrelenmesi de önemsizdir.
- PCI veri güvenliği standart gereksinim 3 altında, cardholder verisini korumalısınız. PCI DSS uyumluluğu, kredi kartı dağıtımıyla ilgilenen herkesin yada tüccarların 2008'den itibaren uyma zorunluluğu olacaktır. En iyi pratik asla gereksiz veriyi depolama, manyetik şerit bilgisi veya birincil hesap numarası (PAN, bir diğer deyişle bilindiği adıyla kredi kartı numarası) . Eğer PAN saklanmışsa, DSS uyumluluk gereksinimleri işaret olamlıdır. Örnek olarak, hangi koşul altında olursa olsun CVV numarasını asla depolayamazsın. Daha fazla bilgi için, lütfen PCI DSS Guideline'ına ve gerekli olan kontrolleri yerine getirin.

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1664
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1101

REFERANSLAR

- CWE: CWE-311 (Failure to encrypt data), CWE-326 (Weak Encryption), CWE-321 (Use of hard-coded cryptographic key), CWE-325 (Missing Required Cryptographic Step), others.
- WASC Threat Classification: No explicit mapping
- OWASP, http://www.owasp.org/index.php/Cryptography
- OWASP Guide, http://www.owasp.org/index.php/Guide to Cryptography
- OWASP, http://www.owasp.org/index.php/Insecure Storage



- OWASP, http://www.owasp.org/index.php/How to protect sensitive data in URL's
- PCI Data Security Standard v1.1,
 https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
- Bruce Schneier, http://www.schneier.com/
- CryptoAPI Next Generation, http://msdn2.microsoft.com/en-us/library/aa376210.aspx

A9- GÜVENSİZ İLETİŞİMLER

Uygulamalar sıklıkla hassas bağlantıların korunması gerektiğinde ağ trafiğini şifrelemede başarısız olurlar. Şifreleme (genellikle SSL) bütün bağlantıların kimlik doğrulaması için özellikle hem internet erişimi olan web sayfaları için hem de sunucu uygulamarı için kullanılmak zorundadır. Diğer bir deyişle, uygulama ya kimlik doğrulaması yada oturum şifrelemeye mecbur bırakılacaktır. Ek olarak, şifreleme hassas veri için her zaman kullanılmalıdır. Kredi kartı veya sağlık bilgileri aktarıldığı zamanlar gibi.

PCI standartları tüm kredi kartı bilgilerinin şifrelenerek internet üzerinden gönderilmesini gerektirir.

ETKİLENEN ORTAMLAR

Bütün web uygulama yapıları güvensiz iletişime karşı savunmasızdır.

SALDIRIYA AÇIKLIK

Hassas veri iletişimini şifreleme hatasından kasıt, ağ trafiğini koklayan bir saldırganın hassas verilerin iletiminden ya da delilleri içeren konuşmalardan verilere ulaşabileceğidir. Göz önünde bulundurulması gereken bir diğer şey ise, farklı ağlar hattın koklanmasına karşı daha az yada fazla duyarlıdır. Ancak, her ağda tehlike oluşturabilecek bir sunucu sayesinde saldırganlar kolayca sistemi ele geçirmek amacıyla sunucuya koklayıcı (sniffer) kuracak ve trafiği dinleyebileceklerdir.

Son kullanıcı ile aranızda SSL bağlantısı kurmanız son derece önemlidir. Muhtemelen onlar uygulamalara erişmek için güvensiz ağları kullanacaklardır. Çünkü HTTP her tekil istekte ya oturum açma ya da kimlik doğrulaması isteyecektir. Doğrulanmış (onaylanmış), login isteklerinde olduğunun aksine bütün trafik SSL üzerinden gidecektir.

Sunucu uygulamalarıyla şifrelenmiş iletişim de ayrıca önemlidir. Muhtemelen bu ağların daha güvenli olmasına rağmen onların taşıdığı bilgi ve veriler çok daha hassas ve çok daha pahalıdır. Bu sebeple, sunucu uygulamalarında SSL kullanımı oldukça önemlidir.

GÜVENLİĞİ DOĞRULAMA

Amaç uygulamanın bütün kimlik doğrulaması yapılmış ve hassas bağlantıların düzgünce şifrelendiğini onaylamaktır.

Otomatik Yapılan Yaklaşımlar: Zayıflık tarama araçları ilk başlangıçta SSL kullanımı doğrulayabilir ve birçok SSL'le ilintili hata bulabilir. Ancak, bu araçlar sunucu uygulama yazılımlarına erişimi olmaz ve onların güvenli olup olmadığını kontrol edemeyebilir. Statik analiz araçları bazı sunucu uygulama çağrılarının analizi ile yardımcı olabilir. Ancak muhtemelen tüm sistem türlerinin özel mantıksal istekleri anlaşılamayacaktır.

Elle Yapılan Yaklaşımlar: Test yapmak SSL'in kullanılabildiğini ve SSL'den kaynaklanan program hatalarını doğrulayabilmektir. Ancak otomatik yapılan yaklaşımlar muhtemelen daha etkili bir yöntemdir. Kodu gözden geçirme bütün sunucu uygulamalarının düzgün ve oldukça etkin SSL kullanımını sağlar.



KORUMA

En önemli koruma kullanıcı doğrulaması gerektiren bağlantılarda ve önemli sayılabilecek verilerin transferi sırasında SSL kullanımıdır. Web uygulamasının düzgünce çalışması için SSL'in yapılandırmasını içeren birçok detay vardır. Bu nedenle kendi çalışma alanınızı anlamak ve analiz etmek önemlidir. Örnek olarak IE 7.0 yüksek derecede güvenilir SSL sertifikalarını yeşil bir barla göstermektedir. Ancak bu kriptografiyi tek başına güvenli kullanmak için uygun bir kontrol değildir.

- Bütün hassas veri içeren transferlerde yada kullanıcı doğrulaması gerektiren bağlantılarda SSL kullanın. Kimlik, kredi kartı ve sağlık bilgileri gibi...
- Bütün sunucu dışı bağlantıların yani web sunucu ile veritabanı sunucusu arasında taşıma katmanının güvenlik önlemlerinin alındığından emin olun.
- PCI Data Security Standart ihtiyaçları 4'ün altında, geçiş sırasında kart sahibinin verilerini korumak zorundasınız. PCI DSS uyumu kredi kartlarıyla herhangi birinin para çekmesi aşamasında 2008 yılında zorunlu hale gelecektir. Genel olarak, müşteri, ortak, personel ve yöneticilerin online erişim sistemleri mutlaka şifrelenmiş SSL erişimi ile yada benzer yöntemlerle olmalıdır. Daha fazla bilgi için, PCI DSS yönetmeliğini inceleyiniz.

ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6430
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4704
- http://www.schneier.com/blog/archives/2005/10/scandinavian_at_1.html

REFERANSLAR

- CWE: CWE-311 (Failure to encrypt data), CWE-326 (Weak Encryption), CWE-321 (Use of hard-coded cryptographic key), CWE-325 (Missing Required Cryptographic Step), others.
- WASC Threat Classification: No explicit mapping
- OWASP Testing Guide, Testing for SSL / TLS, https://www.owasp.org/index.php/Testing for SSL-TLS
- OWASP Guide, http://www.owasp.org/index.php/Guide to Cryptography
- Foundstone SSL Digger,
 http://www.foundstone.com/index.htm?subnav=services/navigation.htm&subcontent=/services/overview-s3i-des.htm

OWASP İlk 10 2007

- NIST, SP 800–52 Guidelines for the selection and use of transport layer security (TLS) Implementations, http://csrc.nist.gov/publications/nistpubs/800-52/SP800-52.pdf
- NIST SP 800–95 Guide to secure web services, http://csrc.nist.gov/publications/drafts.html#sp800-95



A10- URL ERİŞİMİNİ KISITLAMADA BOZUKLUK

Sıklıkla, bir URL'in tek koruması yetkisiz girişlere izin verilmeyen bir sayfaya yönlendirmedir. Ancak, motive edilmiş, yetenekli yada gerçekten şanslı saldırganlar bu adreslere erişebilir, fonksiyonları çağırabilir veya verileri görüntüleyebilir. Belirsiz güvenlik hassas fonksiyonları ve uygulama içerisindeki verileri korumak için yeterli değildir. Hassas fonksiyonlara erişimden önce erişim kontrolü mutlaka yapılmalıdır ki kullanıcının o fonksiyona erişim yetkisi olduğu kontrolü yapılabilsin.

ETKILENEN ORTAMLAR

Bütün web uygulama çerçevelerinin (framework), URL erişimlerini sınırlayamamak konusunda zayıflık vardır.

ZAYIFLIK

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 ile uygulama çevrelenebilir. Uygulamalar sıklıkla codebase tarafından kod kontrol erişimine, kodu yaymak ve açmak için izin verir. Karmaşık bir modeli sonuçlandırmak güvenlik uzmanları gibi geliştiriciler için de anlamayı zorlaştırır.

Bu kusuru içeren bazı yaygın örnekler:

- "Gizlenmiş" veya "Özel" URL'ler, sadece sunum katmanındaki yetkilendirilmiş kullanıcı ve yöneticilerin işleyebilirdi. Ancak bunun varlığını bilen diğer kullanıcılar tarafından da /admin/adduser.php veya /aproveTranser.do gibi sayfalara erişilebilir. Buna özellikle en çok menü kodlarında rastlanılır.
- Uygulamalar genellikle "Gizli" dosyalara erişime izin verir, statik XML yada sistemin ürettiği raporlar, güvenilir güvenlik uygulamalarının gizledikleri gibi.
- Erişim kontrol tedbirini güçlendiren koddur. Ancak eski veya yetersiz olabilir . Örnek olarak, /approveTransfer.do kodunun tüm kullanıcılar tarafından erişilebildiğini düşünün, yetkisi olmayan kullanıcılar bu kod ile yetkisiz işlemler yapabilir.
- Sunucu üzerinde değil ama istemci üzerinde yetkilendirebilen kod, <u>attack on MacWorld</u>
 2007'de olduğu gibi, sunucu aksine web tarayıcısı üzerinden javascript kodu ile 1700\$
 değerinde "Platinium" kullanıcı geçişi onaylanmıştır.

GÜVENLIK DOĞRULAMA

Amaç sunum katmanında yada uygulamadaki tüm URL'ler içinde erişim kontrolü onayı güçlendirilmelidir.

Otomatik Yapılan Yaklaşımlar: zayıflık tarama sistemleri ve statik analiz araçlarının her ikisi de URL erişim kontrolünü onaylama ile ilgili sorunlara sahiptir. Ancak farklı sebeplerden dolayı bu sıkıntılar vardır. Zayıflık tarama programlarının gizlenmiş sayfaları tahmin etme ile herkese açık sayfaları

birbirinden ayırmada bazı sıkıntıları vardır. Statik analiz araçları özel erişim kontrollerini uğraşarak bulmaya çalışır.

Elle Yapılan Yaklaşımlar: En etkili ve gerçek yaklaşım kod inceleme ve güvenlik testlerinin kombinasyonunu kullanmaktır. Eğer mekanizma dışardan güçlendirildiyse yapılandırma incelenmeli ve test edilmelidir.

KORUMA

Uygulamanın fonksiyon ve rol haritasını matris oluşturarak zaman alan yetkilendirmesi, yasaklanmamış URL erişimlerine karşın başarılı korumada anahtar adımdır. Web uygulamalarının her URL üzerinde erişim kontrolü güçlendirilmek zorundadır. Business logic korunmasız bırakılıp ve sunum katmanı içine erişim kontrolü koymak yeterli değildir. Ayrıca işlem boyunca kullanıcının bir kere yetki kontrolünün yapılması da yeterli değildir. Aksi takdirde, saldırgan yetkilendirmenin kontrol edildiği bölgeyi geçip saldırıya diğer aşamalardan gerekli parametreleri göndererek devam diğer aşamalara devam edebilir.

URL erişim kontrolünü açmak dikkatlı planlamayı gerektirir. En önemli görüşler arasında şunlar vardır:

- Erişim kontrol matrislerinin; işinizin, mimarinin ve uygulama dizaynının bir parçası olmasını sağlayın.
- Tüm URL ve iş fonksiyonlarının güvenliğini sağlamak efektif erişim kontrolü mekanizması tarafından korunduğundan emin olun. Bu ise herhangi bir işlem gerçekleşmeden önce, kullanıcı rolü ve yetkilendirmesini doğrulamak demektir. Bu işlemin her aşamada tekrarlandığından emin olun.
- Uygulama konuşlanmadan önce **saldırı testi uygulayın.** Böylece saldırıya motive edilmiş becerili saldırgan tarafından uygulamanın kötüye kullanılması engellenecektir.
- Özellikle çalıştırılabilir uzantılara sahipse .php gibi **include/library dosyalarını** çok yakından takip edin. Bunları daha güvenli olabilecekleri web ana dizininden uzakta tutun. Onlara direkt olarak erişilemeyeceğini kontrol edin. Direkt olarak erişilemeyeceğini kontrol edin ve sadece sistem kütüphanelerinin çağırabilmesini sağlayın.
- Kullanıcıların, özel veya gizli URL'lerin veya API'lerin farkında olmayacağını sanmayın. Daima yönetici ve yetkileri yüksek öncelikli işlerin korunduğundan emin olunmalıdır.
- 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 direkt 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.



ÖRNEKLER

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0147
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0131
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1227

REFERANSLAR

- CWE: CWE-325 (Direct Request), CWE-288 (Authentication Bypass by Alternate Path),
 CWE-285 (Missing or Inconsistent Access Control)
- WASC Threat Classification:
 http://www.webappsec.org/projects/threat/classes/predictable-resource-location.shtm
- OWASP, http://www.owasp.org/index.php/Forced browsing
- OWASP Guide, http://www.owasp.org/index.php/Guide to Authorization

BURADAN NEREYE DEVAM ETMELİ

OWASP Top 10 dokümanı sizin web uygulama güvenliği seyahatinizin sadece başlangıç noktasıdır.

Dünya üzerindeki 6 milyar insan ikiye bölünebilir: birinci grup, bütün iyi yazılım şirketlerinin bilinen hatalarla ürünlerini gönderdiğini bilenlerdir. İkinci grup ise bunu bilmeyenlerdir.

Eric Sink, Guardian 15 Mayıs 2006

Sizin kullanıcılarınızın yada müşterilerinizin birçoğu ikinci gruptadır. Bu sorunun sizin için genel olarak web uygulama güvenliğinin durumunu ve kodunuzu geliştirmek için bulduğunuz fırsat olduğunu nasıl bilebilirler. Her yıl milyarca dolar ve milyonlarca insanın birçoğu bu dokümanda tartışılan zayıflıklar yüzünden kimlikleri çalınıyor ve bunların cezasını çekiyorlar.

TASARIMCILAR IÇIN

Düzgün bir şekilde uygulamalarınızın güvenliğini sağlamak için neyin güvenliğini sağladığınızı (değer sınıflandırması) bilmeniz gerekiyor. Geleneksel olmayan her hangi bir dizayn uygulaması iyi dozda bir güvenlik gerektirir.

- Risk seviyesi modeline ve güvenlik sınıflandırmasına uygun "yeterli" güvenlik
 uyguladığınızdan emin olun. Ancak, yükü (SOX, HIPAA, Sasel gibi) kurallara bağlı kalarak
 hafifletin. O bugünün minimum memnuniyetinden daha fazla kaynak ve zamana ihtiyacı
 olması için sahiplenecektir.
- İş ihtiyaçları ile ilgili olarak sorular sorun, özellikle fonksiyonel olmayan gözden kaçan ihtiyaçlar için.
- OWASP Secure Software Contract Annex aracılığıyla müşterilerinizle çalışın.
- **Daha güvenli bir tasarımı destekleyin** tehdit modelini (kitap referansları içindeki [HOW1] 'e bakınız) kullanarak basit ama savunması daha kolay tasarımları içersin.
- Güvenli, herkesin gözü önünde olmayan, sağlam olduğunu göz önünde bulundurduğunuzdan emin olun.
- Tasarımınızın güvenlik planı ve standartlar ile uyumlu olduğundan emin olun. COBIT yada PCI DSS 1.1 standartlar gibi.

GELİŞTİRİCİLER İÇİN

Birçok geliştirici hali hazırda temel web uygulama güvenliği üzerine iyi bir bilgi sahibidir. Daha ileri seviyede daha efektif bir web uygulama güvenliği pratik gerektirmektedir. Herhangi biri zarar verebilir. (etkili bir içeri sızma (penetration) testi gibi) – Güvenli bir yazılım inşa etmek ustalık ister. Usta olma amacını güdün.

OWASP' a katılımı yada yerel grup toplantılarına katılmayı düşünün.



- Eğer eğitim için bütçeniz varsa **güvenli kod yazma için eğitim arayın.** Eğer eğitim için ayrılmış bütçeniz yoksa yöneticilerinizle bu konuyu görüşün.
- Geleceğinizi güvenli bir şekilde tasarlayın Tasarımda basitliği ve derin savunmayı düşünün
- Standart kodlamayı benimseyin. Bu sizi daha güvenli kodlamaya yönlendirir.
- Var olan kodu yeniden daha güvenli teknikleri kullanarak seçtiğiniz platformda yeniden oluşturun,
- <u>OWASP Rehberini</u> gözden geçirerek kodunuza seçilen kontrolleri uygulamaya başlayın. Birçok güvenlik rehberinden farklı olarak, o güvenli yazılım oluşturmayı size yardım etmek için tasarladı.
- Kodunuzu güvenlik sorunlarına karşın test edin. Bunu kendinizde alışkanlık haline getirin.
- Sizin ortamınıza uygun olanları görmek için **Kitabın referanslarını gözden geçirin**

AÇIK KAYNAK KODLU PROJELER İÇİN

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.

- OWASP' a katılımı yada yerel grup toplantılarına katılmayı düşünün.
- Eğer projeniz 4'den fazla geliştiriciye sahipse, en azından bir geliştiriciyi güvenlikten sorumlu olarak atamayı düşünün,
- Özelliklerinizi güvenli olarak tasarlayın derinliğine güvenlik ve tasarımda basitlik kavramlarını göz önünde tutun,
- Daha güvenli kod yapılarını özendiren kodlama standartlarını kabul edin,
- Güvenlik kusurlarının düzgün olarak ele alınmasını sağlayan, sorumluyu söyleme politikasını kabul edin,
- Sizin ortamınıza uygun olanları görmek için **Kitabın referanslarını gözden geçirin**

UYGULAMA SAHİPLERİ İÇİN

Ticari hayatın içindeki uygulama sahipleri zaman ve kaynak konusunda sıkıntılıdır. Uygulama sahiplerinin şunları yapmalıdır:

- Yazılım üreticilerle <u>OWASP Secure Software Contract Annex</u> vasıtasıyla çalışmalı
- İş ortamının istekleri fonksiyonel ihtiyaçlar olmadığından (NFRs) güvenlik ihtiyaçları gibi emin olsunlar.

OWASP İlk 10 2007

- Varsayılan özellik olarak güvenliği içeren tasarımları destekleyin, tasarımda basitlik savunmada derinlik sağlarlar.
- Geliştiriciler içinde Güçlü güvenlik bilgi birikimine sahip personel alın yada personelinizi bu şekilde eğitin.
- Projenin başından sonuna **güvenlik sorunlarına karşın testler yapın:** tasarım, test ve dağıtım aşamalarında.
- Güvenlik amaçlı projelerde kaynak, bütçe ve zamana izin verin.

C-SEVİYE YÖNETİM İÇİN

Organizasyonunuz güvenli geliştirme yaşam döngüsüne (SDLC) sahip olmak zorundadır. Zayıflıklar ürününüzü piyasaya sürdükten sonra değil geliştirme aşamasında daha ucuza çözüm bulabilir. Makul bir SDLC sadece İlk 10 testleri içermez aynı zamanda şunları da içerir:

- Ödeme poliçesi ve kontratının güvenlik ihtiyaçlarını da içerdiğinden emin olun.
- Özel kodlar için, sizin poliçe ve standartlarınızdaki güvenli kodlama prensiplerini benimseyin.
- **Geliştiricilerinizi güvenli kodlama teknikleri hakkında yetiştirin** ve bu yeteneklerinin geliştiğinden emin olun.
- Bütçeniz güvenliğe uygun kod analiz araçlarını da içersin
- Yazılım üreticinizi güvenliğinizin alt sınırının önemini hakkında uyarın.
- Mimarlarınızı, tasarımcılarınızı ve iş yeri personelinizi web uygulama güvenliği temelleri hakkında eğitin.
- Kod denetleyici olarak bağımsız düşünebilecek üçüncül kişileri düşünün
- Sorumluyu ortaya çıkarma çalışmalarını benimseyin ve düzgün bir süreç oluşturacak zayıflık raporlama sistemini kendi ürünleriniz için oluşturun.



REFERANSLAR

OWASP PROJELERI

OWASP web uygulama güvenliği için birinci sitedir. <u>OWASP sitesi</u> birçok <u>projeyi</u>, <u>forumu</u>, <u>blog</u>u, <u>sunumu</u>, <u>aracı</u> ve <u>dokümanı</u> barındırmaktadır. OWASP her yıl iki ana <u>web uygulama güvenliği konferansına</u> ve 80 üzerinde yerel <u>bölüme</u> sahiplik yapar.

Aşağıdaki OWASP projeleri büyük ihtimalle kullanışlı olacaktır.

- OWASP Guide to Building Secure Web Applications
- OWASP Testing Guide
- OWASP Code Review Project (geliştirilme aşamasında)
- OWASP PHP Project (geliştirilme aşamasında)
- OWASP Java Project
- OWASP .NET Project

KITAPLAR

Zorunluluk nedeniyle bu liste çok uzun tutulamamıştır. Bu referansları kullanarak bazı başlıklarda kendi bölgenizdeki kitapçılarda istediğiniz kitapları bulma şansınız var.

- [ALS1] Alshanetsky, I. "php|architect's Guide to PHP Security", ISBN 0973862106
- [BAI1] Baier, D., "Developing more secure ASP.NET 2.0 Applications", ISBN 978-0-7356-2331-6
- [GAL1] Gallagher T., Landauer L., Jeffries B., "Hunting Security Bugs", Microsoft Press, ISBN 073562187X
- [GRO1] Fogie, Grossman, Hansen, Rager, "Cross Site Scripting Attacks: XSS Exploits and Defense", ISBN 1597491543
- [HOW1] Howard M., Lipner S., "The Security Development Lifecycle", Microsoft Press, ISBN 0735622140
- [SCH1] Schneier B., "Practical Cryptography", Wiley, ISBN 047122894X
- [SHI1] Shiflett, C., "Essential PHP Security", ISBN 059600656X
- [WYS1] Wysopal et al, The Art of Software Security Testing: Identifying Software Security Flaws, ISBN 0321304861

WEB SITELERI

- OWASP, http://www.owasp.org
- MITRE, Common Weakness Enumeration Vulnerability Trends, http://cwe.mitre.org/documents/vuln-trends.html
- Web Application Security Consortium, http://www.webappsec.org/
- SANS Top 20, http://www.sans.org/top20/
- PCI Security Standards Council, publishers of the PCI standards, relevant to all organizations processing or holding credit card data, https://www.pcisecuritystandards.org/
- PCI DSS v1.1, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
- Build Security In, US CERT, https://buildsecurityin.us-cert.gov/daisy/bsi/home.html



MİNİ SÖZLÜK

buffer	arabellek
communication	iletişim
Cross Site Request Forgery	siteler ötesi istek sahteciliği
Cross site Scripting	siteler arası betik yazma
Cryptographic	Kriptografik
denial of service	hizmet dışı bırakma
error handling	hata işleme
exploit	gerçeklenmiş açık
Failure to Restrict URL Access	URL Erişimini kısıtlamada bozukluk
flaw	tasarım hatası
framework	çerçeve
Information leakage and Improper Error HandBilgi sızıntısı ve Uygunsuz hata işleme	
injection	enjeksiyon
Injection flaws	Enjeksiyon tasarım hatası
Insecure Communications	Güvensiz iletisimler
Insecure Cryptographic Storage	Güvensiz Kriptografik Depolama
Insecure Direct Object Reference	Güvensiz Doğrudan Nesne Başvurusu
leakage	sızıntı
Malicious File Execution	zararlı dosya çalıştırma
phishing	parola balıkçılığı
script	betik
session	oturum
token	jeton
top 10	ilk 10
vulnerability	zayıflık

Tercümede kullanılan sözlükler:

- http://www.webguvenligi.org/sozluk/
- http://www.tureng.com/
- http://www.zargan.com
- http://www.tbd.org.tr/genel/sozluk.php