Testing for Local File Inclusion (Yerel Dosya Ekleme Testi)

Summary (Özet)

Dosya Kapsayıcılığı güvenlik açığı, bir saldırganın bir dosya içermesine ve genellikle hedef uygulamada uygulanan bir “dinamik dosya dahil etme” mekanizmalarından yararlanmasına izin verir. Güvenlik açığı, uygun doğrulama olmadan kullanıcı tarafından sağlanan girdilerin
kullanılması nedeniyle ortaya çıkar.

Bu, dosyanın içeriğini çıkarmak gibi bir şeye yol açabilir, ancak ciddiyetine bağlı olarak aşağıdakilere de yol açabilir:

- Web sunucusunda kod yürütme
- Şarım gibi istemci tarafında kod uygulaması, çapraz site komut dosyası (XSS) gibi diğer saldırılara yol açabilir
- Hizmetin Reddi (DoS)
- Hassas Bilgi Açıklaması

Yerel dosya dahil etme (LFI olarak da bilinir), uygulamada uygulanan savunmasız kapsayıcılık prosedürlerinin istismarı yoluyla sunucuda zaten yerel olarak bulunan dosyaları dahil etme işlemidir. Bu güvenlik açığı, örneğin, bir sayfanın giriş olarak, dahil edilmesi gereken dosyaya giden yolu aldığında ve bu girdinin düzgün bir şekilde sterilize edilmediği ve dizin geçiş karakterlerinin (nokta-nokta-slash gibi) enjekte edilmesine izin verdiğinde ortaya çıkar. Çoğu örnek savunmasız PHP komut dosyalarına işaret etse de, JSP, ASP ve diğerleri gibi diğer teknolojilerde de yaygın olduğunu akılda tutmalıyız.

How To Test (Nasıl Test Edilir)

LFI, yollar geçtiğinde ortaya çıktığında ortaya çıktığından `include` İtiraflar düzgün bir şekilde sterilize edilmemiştir, bir kara kutu test yaklaşımında, dosya adlarını parametre olarak alan komut dosyalarını aramalıyız.

Aşağıdaki örneği göz önünde bulundurun:

`http://vulnerable_host/preview.php?file=example.html`

Burası LFI için denemek için mükemmel bir yer gibi görünüyor. Bir saldırgan yeterince şanslıysa ve diziden uygun sayfayı kendi adıyla seçmek yerine, komut dosyası doğrudan giriş parametresini içerir, sunucuya keyfi dosyaların dahil edilmesi mümkündür.

Tipik bir kavram kanıtı, şifreli dosyayı yüklemek olacaktır:

`http://vulnerable_host/preview.php?file=../../../../etc/passwd`

Yukarıda belirtilen koşullar yerine getirilirse, bir saldırgan aşağıdaki gibi bir şey görür:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
alex:x:500:500:alex:/home/alex:/bin/bash
margo:x:501:501::/home/margo:/bin/bash
...
```

Böyle bir güvenlik açığı varken bile, sömürüsü gerçek yaşam senaryolarında daha karmaşık olabilir. Aşağıdaki kod parçasını göz önünde bulundurun:

`<?php include($_GET['file'].".php"); ?>`

Rastgele bir dosya adı olan basit ikame, postfix olarak çalışmaz `.php` Sağlanan girişe eklenir. Onu atlamak için, bir testçi beklenen sömürüyü elde etmek için birkaç teknik kullanabilir.

Null Byte Injection (Null Byte Enjeksiyonu)

The (İngilizce) `null character` (Ayrıca da bilinir) `null terminator` ya da `null byte`),
bir ipin ucunu işaretlemek için ayrılmış bir karakter olarak kullanılan birçok karakter setinde mevcut olan sıfır değerine sahip bir kontrol karakteridir. Kullanıldıktan sonra, bu özel bayttan sonraki herhangi bir karakter göz ardı edilecektir. Yaygın olarak bu karakteri enjekte etmenin yolu URL kodlanmış dize ile olacaktır.
`%00` İstenen yola ekleyerek. Önceki örneğimizde, bir talepte bulunmak için `http://vulnerable_host/preview.php?file=../../../../etc/passwd%00` The The'i görmezden gelirdi

`.php` Giriş dosya adına uzantı eklenir ve başarılı bir sömürünün bir sonucu olarak bir saldırgana temel kullanıcıların bir listesini döndürür.

Path and Dot Truncation (Yol ve Nokta Gövdesi)

PHP kurulumunun çoğu dosya adı 4096 bayttır. Herhangi bir dosya adı bu uzunluktan daha uzunsa, PHP sadece herhangi bir ek karakteri atarak kesilir. Bu davranışı kötüye kullanmak, PHP motorunun görmezden gelmesini mümkün kılar `.php` 4096 bayt sınırından çıkarak uzatma. Bu olduğunda, hiçbir hata tetiklenmez; ek karakterler basitçe düşürülür ve PHP normal şekilde yürütülmesine devam eder.

Bu baypas genellikle, dosya yolunun bir kısmını Unicode kodlaması, çift kodlamanın getirilmesi veya geçerli istenen dosya adını temsil edecek başka herhangi bir girdi gibi diğer mantık bypass stratejileri ile birleştirilecektir.

PHP Wrappers (PHP Wrappers)

Yerel Dosya Kapsayıcılığı güvenlik açıkları, yalnızca bir saldırganın savunmasız uygulamayı barındıran sunucudan hassas verileri okumak için kullanabileceği güvenlik açıkları olarak görülür. Bununla birlikte, bazı özel uygulamalarda bu güvenlik açığı, saldırıyı LFI'den uzaktan Kod Yürütme güvenlik açıklarına yükseltmek için kullanılabilir.

Bu geliştirme, bir saldırganın LFI güvenlik açığını belirli PHP ambalajlarıyla birleştirebildiğinde yaygındır.

Bir ambalaj, bazı ek işlevler gerçekleştirmek için diğer kodu çevreleyen bir koddur. PHP, dosya sistemi işlevleri ile kullanılmak üzere birçok yerleşik ambalaj uygular. Bir uygulamanın test sürecinde kullanımları tespit edildikten sonra, tespit edilen zayıflığın gerçek riskini tanımlamak için kötüye kullanmaya çalışmak iyi bir uygulamadır. Aşağıda, en sık kullanılan ambalajlara sahip bir liste alabilirsiniz, ancak bunun kapsamlı olmadığını düşünmeniz ve aynı zamanda hedef tarafından istihdam edilirse daha derin bir ad hoc analizi gerektirecek özel ambalajları kaydetmeniz mümkün.

PHP Filter (PHP Filtresi)

Yerel dosya sistemine erişmek için kullanılır; Bu, bir dosyanın açıldığı sırada bir akışa filtre uygulama yeteneğini sağlayan duyarsız bir ambalajdır. Bu ambalaj, sunucunun yürütmesini engelleyen bir dosyanın içeriğini elde etmek için kullanılabilir. Örneğin, bir saldırganın PHP dosyalarının içeriğini okumasına izin vererek, kimlik bilgileri veya diğer istismar edilebilir güvenlik açıkları gibi hassas bilgileri tanımlamak için kaynak kodu almak için.

Sarma makinesi şöyle kullanılabilir `php://filter/convert.base64-encoderesource=FILE` Nerede `FILE` Alacak dosyadır. Bu yürütmenin kullanılması sonucunda, hedef dosyanın içeriği okunacak, taban 64'e kodlanacak (bu, yürütme sunucusu tarafını engelleyen adımdır) ve Kullanıcı-Ajan'a iade edilecektir.

PHP ZIP (PHP ZIP)

PHP 7.2.0'da, `zip://` Sarıcı manipüle etmek için tanıtıldı `zip` Sıkıştırılmış dosyalar. Bu ambalaj aşağıdaki parametre yapısını bekler: `zip:///filename_path#internal_filename` Nerede `filename_path` Kötü amaçlı dosyaya giden yol ve `internal_filename` Kötü amaçlı dosyanın işlenen ZIP dosyasının içine yerleştirildiği yoldur. Sömürü sırasında, neyin yaygın olduğudur. `#` URL Kodlanmış değeri ile kodlanır `%23` . .

Bu ambalajın kötüye kullanılması, bir saldırganın sunucuya yüklenebilecek kötü amaçlı bir ZIP dosyası tasarlamasına izin verebilir, örneğin bir avatar resmi olarak veya hedef web sitesinde bulunan herhangi bir dosya yükleme sistemi kullanarak (otornak) `php:zip://` Wrapper, zip dosyasının herhangi bir özel uzantıya sahip olmasını gerektirmez) LFI güvenlik açığı tarafından yürütülür.

Bu güvenlik açığını test etmek için, sağlanan önceki kod örneğine saldırmak için aşağıdaki prosedür takip edilebilir.

1. İdam edilecek PHP dosyasını oluşturun, örneğin içerikle `<?php phpinfo(); ?>` ve onu olduğu gibi kaydedin `code.php`
2. İlgili yeni bir ZIP dosyası olarak sıkıştırın `target.zip`
3. Yeniden adlandırın `target.zip` Dosyaya dosya `target.jpg` Uzatma doğrulamasını atlamak ve avatar resminiz olarak hedef web sitesine yüklemek.
4. Bunu desteklemek `target.jpg` dosya yerel olarak sunucuda depolanır `../avatar/target.jpg` Yol, aşağıdaki yükü savunmasız URL'ye enjekte ederek PHP ZIP ambalajı ile güvenlik açığından yararlanın: `zip://../avatar/target.jpg%23code` (Bunu unutmayın `%23` Rakiplere karşılık gelir `#`).

Bizim örneğimizden beri `.php` Yükümüzle uzatma, talep için uygundur `http://vulnerable_host/preview.php?file=zip://../avatar/target.jpg%23code` İdam edilmesiyle sonuçlanacak `code.php` Kötü amaçlı ZIP dosyasında mevcuttur.

PHP Data (PHP Verileri)

PHP 5.2.0'dan beri mevcut olan bu sarmalayıcı aşağıdaki kullanımı beklenmektedir: `data:/text/plain;base64,BASE64_STR` Nerede `BASE64_STR` Dosyanın işlemlenilecek Base64 kodlu içeriği olması bekleniyor. Bu ambalajın yalnızca seçenek varsa kullanılabilir olacağını düşünmek önemlidir. `allow_url_include` Etkin olurdu.

Bu ambalajı kullanarak LFI'yi test etmek için, yürütülecek kod, örneğin, uygulanacak kodlanmış olarak, `<?php phpinfo(); ?>` Kod şu şekilde kodlanacaktır: `PD9waHAgcGhwaW5mbygpOyA/Pg==` Böylece yük şu şekilde sonuçlanacaktır: `data://textplain;base64,PD9waHAgcGhwaW5mbygpOyA/Pg==` . .

PHP Expect (PHP Bekleyin)

Varsayılan olarak etkin olmayan bu sarmalayıcı, süreçlere erişim sağlar `stdio` , `stdout` ve `stderr` . . Kullanılmayı beklemek `expect://command` Sunucu, sağlanan komutu üzerinde yürütür. `BASH` Ve sonuç olarak iade edin.

Remediation (Düzeltme)

Dosya dahil etme güvenlik açıklarını ortadan kaldırmak için en etkili çözüm, kullanıcı tarafından gönderilen girdileri herhangi bir dosya sistemi / çerçeve API'ye geçirmekten kaçınmaktır. Bu mümkün değilse, uygulama dosyanın izinli bir listesini koruyabilir, bu sayfaya dahil edilebilir ve daha sonra seçilen dosyaya erişmek için bir tanımlayıcı (örneğin endeks numarası) kullanabilir. Geçersiz bir tanımlayıcı içeren herhangi bir talep reddedilmelidir, bu şekilde kötü niyetli kullanıcıların yolu manipüle etmeleri için saldırı yüzeyi yoktur.

Bu konudaki iyi güvenlik uygulamaları için File Yük Hile Sayfasına göz atın.

Tools (Araçlar)

- kadimus
- LFI Suite
- OWASP Zed Attack Proxy (ZAP)

References (Referanslar)

- Wikipedia
- Null character
- Unicode Encoding
- Double Encoding
- PHP Supported Protocols and Wrappers
- RFC 2397 - The “data” URL scheme