OWASP TOP-10 ZAFİYETLERİNİN UYGULANMASI

1.SSRF

SSRF (Sunucu Tabanlı İstek Sahteciliği) bu zafiyeti bulunduran sunucu üzerinden, bu sunucu adına dışarıdan herhangi bir web adresine istekte bulunmamızı sağlar. Sunucu adına istek attığımız adres veya endpoint genellikle istenmeyen ve beklenmeyen adreslerdir.

SSRF Zafiyeti OWASP TOP-10 listesinde, 10. Sırada bulunmaktadır.

Örnek olarak bir e-ticaret uygulamasını düşünelim. Bu uygulama bir ürünün stokta olup olmadığını öğrenmek için belirli bir APİ’ye istek atsın. Biz de APİ’ye istek attığı http requestini yakalayıp, APİ kısmında yazan adresi değiştirdikten sonra, yazdığımız adrese istek gitmesini sağlarsak SSRF’i sömürmüş oluruz.

LAB ÇÖZÜMÜ:

Labımı portswigger sitesi üzerinde çözeceğim.

Lab linki:

metin, elektronik donanım, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Labıma giriş yaptığımda bir e-ticaret uygulaması görüyorum. Çeşitli ürünler listelenmiş. Bu sayfayı yenileyip herhangi bir url değerinin girdi alınıp alınmadığını kontrol ediyorum.

metin, ekran görüntüsü, yazılım, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu

İsteğimi inceliyorum herhangi bir yerde url değerinin girdi olarak alınmadığını görüyorum.

Ürünlerin detaylarına girdiğimde en alt kısımda hangi ülkede stoğunun olduğunu sorgulamamız için bir yer görüyorum.

metin, yazı tipi, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Check stock’a tıklıyorum ve isteği burpten inceliyorum.

metin, ekran görüntüsü, yazılım, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu



stockApi kısmında bir url girdisi aldığını görüyorum. Tahminimce stok bilgisini bu url’e http isteği attıktan sonra elde ediyor. Bu stockApi kısmına <http://localhost> yazmayı deniyorum. Eğer bir engelleme, input validation yoksa bana localhost’u response olarak dönmeli.

metin, sayı, numara, yazılım, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Localhost’a attığım http isteği bana 200 olarak döndü. Ve localhost sayfasını döndü. Gelen http response’u inceliyorum.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Localhost altında /admin sayfasının olduğunu görüyorum, <http://localhost/admin> şeklinde isteğimi atıyorum.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

İsteğimi atıyorum ve 200 ok dönüyor. Response’u incelediğim zaman alt kısımlarda /delete adında bir sayfa daha olduğunu ve bu sayfanın’da “username” adında bir parametre aldığını görüyorum. Lab’ımızın amacı bu alt sayfaları kullanarak “Carlos” ismindeki kullanıcıyı silmekti.

<http://localhost/admin/delete?username=carlos> şeklide isteğimi atarak carlos’u silmeyi deniyorum.

metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

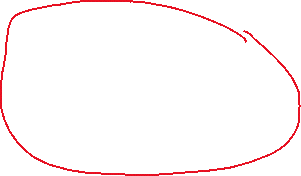
Açıklama otomatik olarak oluşturuldu

İsteğimi attım ve herhangi bir hata almadım. Bu Carlos kullanıcısını başarıyla sildiğim anlamına geliyor.

Kontrol etmek amacıyla <http://localhost/admin> adresine isteğimi atarak kullanıcılara bakıyorum.

metin, sayı, numara, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu



Kullanıcılar kısmında carlos’un olmadığını görüyorum. Yani lab’ı başarıyla tamamladım 😊

Bu labımızda, uygulamanın amacının bir ürünün stokta olup olmadığını

Öğrenmekti. Ama biz uygulama sunucusu üzerinden atılan isteğin adresini değiştirdik ve admin olmayan normal kullanıcıların erişmemesi gereken sayfalara bu sayede eriştik. Ve bütün bunları web sunucusuna http isteği attırarak başardık. Böylece SSRF açığını başarıyla sömürdük.

2.CSRF

CSRF Açığı (Client Side Request Forgery) yani istemci tabanlı istek sahteciliği açığı, OWASP TOP-10 2021 sıralamasında 1. Sırada olan Broken Access Control kategorisine girer.

CSRF açığının ortaya çıkma nedeni temelde HTTP protokolünde herhangi bir doğrulama mekanizması bulunmamasıdır. Bu eksiklik yüzünden web uygulamalarında cookie’ler kullanılır. Cookieler aracılığıyla bir web uygulamasına 2. gidişimizde tekrar giriş yapmamıza gerek kalmaz. Eğer cookieler olmasaydı her seferinde giriş yapmamız gerekecekti.

CSRF açığı kullanıcının oturum açmış olduğu ve session’u bulunduğu web uygulamasına başka bir web adresi, tab üzerinden kullanıcının isteği dışında istek atılmasıdır. Bu istekler kullanıcının parolasını değiştirme gibi kritik işlemler olabilir.

LAB ÇÖZÜMÜ:

Portswigger üzerinden çözeciğim labın linki: <https://portswigger.net/web-security/csrf/lab-no-defenses>

Lab açıklamasını okuduğumda e-mail değiştirme kısmında bir CSRF açığının olduğunu söylemekte.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Verilen kullanıcı bilgileriyle öncelikle lab sayfamda girişimi yaptım. Update email kısmını görüyorum buraya rastgele bir mail yazıp gönderiyorum ve giden isteği inceliyorum.

metin, yazılım, web sayfası, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

İsteği incelediğimde herhangi bir csrf token’i göremiyorum. Origin ve Referer kısımları bulunuyor ama buraları kontrol ettiğini düşünmüyorum. Host kısmını ve /my-account/change-email kısmını alıyorum. Portswigger üzerinden CSRF PoC html’i buluyorum. Ve bu html dosyasını düzenliyorum.

Son haldeki html dosyası aşağıdaki gibi:

metin, multimedya yazılımı, yazılım, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yukarıdaki html dosyasına kurban tıkladığı zaman karşısında hiçbir şey görmeyecek. Ancak gizli bir input değeri bulunduğu ve bu input değerinin value değeri girildiği için, web sitesine tıkladığı anda kullanıcının mail adresi value kısmına girilen mail adresiyle değişmiş olacak.

Kullanıcının bu html dosyasını açmasının simüle edilmesi için portswigger’ın oluşturduğu exploit server’a gidiyorum ve body kısmına oluşturduğum html’i yapıştırıyorum.

metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kurban’a tıklaması gereken linki gönderiyorum.

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu



metin, ekran görüntüsü, web sayfası, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu



Kurbanı gönderdiğim linke tıklatarak mail adresini istediğim şekilde değiştirdim. CSRF saldırısını başarıyla gerçekleştirdim :)

3. OS Command İnjection

OS Command injection açığı OWASP TOP-10 2021 listesinde 3. Sırada bulunan İnjection kategorisine girmektedir.

Bu güvenlik açığı saldırganın web sunucusu üzerinde istediği sistem komutlarını çalıştırmasını sağlar. Bunun anlamı saldırganın sunucuda yetkisi yettiği düzeyde istediğini yapabilmesidir. Saldırgan bu açığı kullanarak çalıştırdığı komutlarla yetki yükseltmesi de yapabilir.

LAB ÇÖZÜMÜ:

PortSwigger üzerinden çözeceğim labın linki: <https://portswigger.net/web-security/os-command-injection/lab-simple>

metin, ekran görüntüsü, web sitesi, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu

Laba giriş yaptım yukarıdaki gibi ürünler sıralanmış. Bu sayfada herhangi bir input alanı göremiyorum. Command injection saldırımı gerçekleştirmem için bir yerden girdi vermeliyim. Ürün detaylarına giriyorum.

Web uygulamasının yapısı gereği url’deki productİd kısmına bir deneme yapmıyorum. Bu productİd üzerinden veritabanına sorguda bulunuyor ve ürün detaylarını getiriyor. Buradan bir Command injection değil SQL injection çıkabilir. Ama şuan amacımız Command İnjection’u sömürmek.

Ürün detaylarında aşağıda stok sorgulama kısmı görüyorum.

metin, insan yüzü, ekran görüntüsü, kişi, şahıs içeren bir resim

Açıklama otomatik olarak oluşturuldu



Check stock butonuna tıklıyorum ve burp üzerinden giden isteği inceliyorum.

metin, web sayfası, yazılım, web sitesi içeren bir resim

Açıklama otomatik olarak oluşturuldu



İsteğimin body kısmında 2 adet parametre gittiğini görüyorum. Burada web uygulamasının bir input aldığını görüyorum. Bu input noktasına denemelerimi yapacağım. Bu stok sorgulamasının nasıl yapıldığını tahmin etmeye çalışıyorum ve aklıma bir senaryo geliyor.

Stokkontrol.py adında bir programımız olsun ve bu program 2 tane parametre alsın. Bu uygulamayı kendi web sunucumda çalıştırmak isteseydim linux komut satırında :

Python3 stokkontrol.py (parametre1) (parametre2) şeklinde çalıştırırdım.

Test ettiğim web sitesinin de bu şekilde stok kontrol yaptığını düşünüp ilk başta buna göre girdi vereceğim. Eğer başarıya ulaşamazsam başka ihtimaller düşüneceğim.

Web uygulaması tahmin ettiğim şekilde aldığı girdileri yerleştiriyorsa, bu girdilere linux komut satırındaki kaçış ifadelerini kullanarak payloadlarımı deniyorum.

metin, yazılım, web sayfası, web sitesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

productİd kısmına ;whoami+# payloadımı giriyorum. Payloadımın başındaki ( ; ) işareti çalıştırılan komuttan dışarı çıkmayı sağlıyor, yani python3 stokkontrol.py komutunu sonlandırarak artık parametre beklemeyi bırakıyor. Daha sonra whoami komutunu çalıştırıyorum ve sonuna ( # ) koyuyorum. Komut satırı #’den sonraki komutları görmezden gelir.

Response’da görüldüğü gibi whoami komutunu başarıyla çalıştırdım. Ve linux sistemi üzerinde OS Command İnjection açığını sömürdüm 😊