CSRF(跨站请求伪造)漏洞总结

一、跨站请求伪造漏洞介绍

1.CSRF介绍

CSRF概念：CSRF为跨站请求伪造(Cross—Site Request Forgery)，跟XSS攻击一样，存在巨大的危害性，你可以这样来理解：

攻击者盗用了你的身份，以你的名义发送恶意请求，对服务器来说这个请求是完全合法的，但是却完成了攻击者所期望的一个操作，比如以你的名义发送邮件、发消息，盗取你的账号，添加系统管理员，甚至于购买商品、虚拟货币转账等。CSRF实际上就是攻击者通过各种方法伪装成目标用户的身份，欺骗服务器，进行一些非法操作，但是这在服务器看来却是合法的操作。

2.CSRF与XSS的区别

尽管CSRF听起来很像XSS，但是却又与XSS有本质的区别。**最本质的区别就是XSS利用的是用户信任服务器，CSRF利用的是服务器信任用户。** XSS是由于用户信任服务器，因此攻击者可以将恶意的JS代码上传到服务器，用户由于信任服务器就会去访问服务器，这样，用户就容易收到XSS攻击；CSRF是由于服务器信任用户，因此攻击者会伪装成用户的身份去进行一些非法操作，但是服务器会认为这是合法的操作，这样用户就受到了CSRF攻击。

二、跨站请求伪造漏洞原理

1.跨站请求伪造原理解释

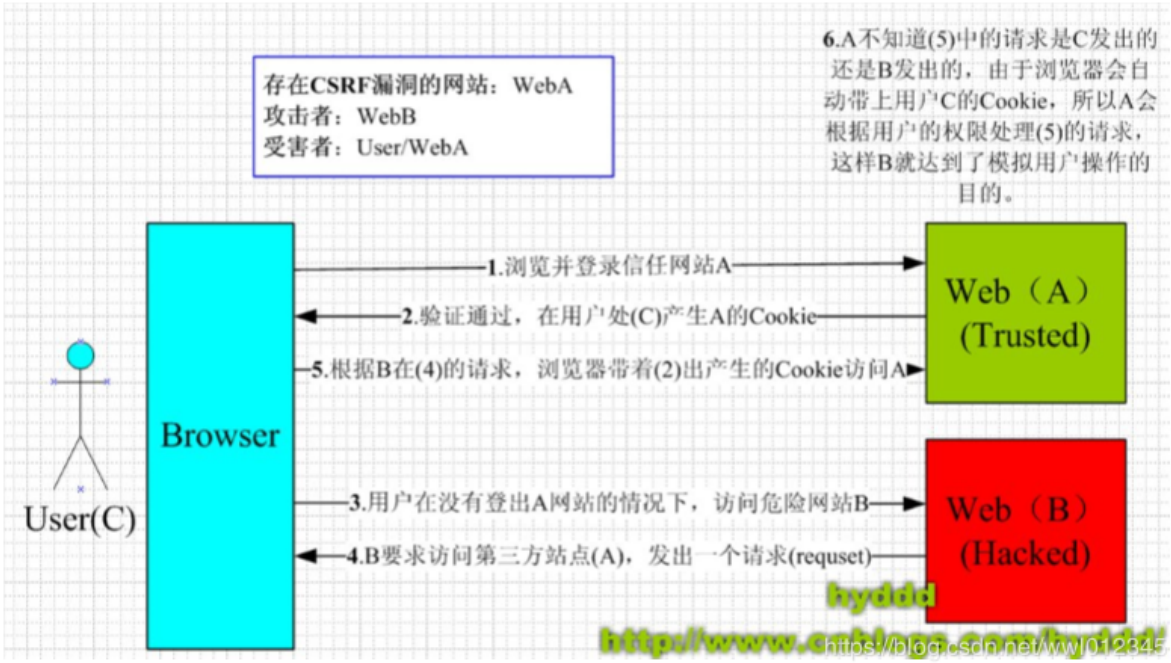
其实CSRF就是攻击者利用目标用户的身份，以目标用户的名义去执行某些非法操作。CSRF能做的事情包括:以目标用户的名义发邮件、发消息、盗取目标用户的的账号，甚至购买商品、虚拟货币转账，这些会泄露个人隐私并涉及目标用户的财产安全。

2.进行CSRF攻击的前提条件

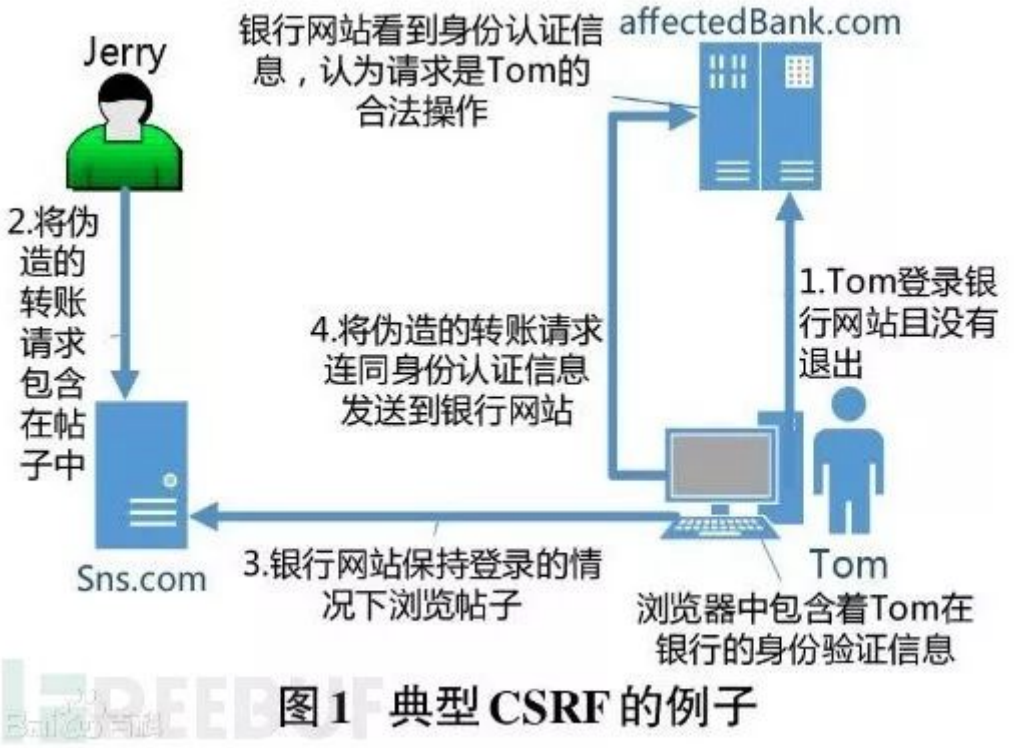
- 目标用户已经登录了网站，并且能够执行网站的功能
- 目标用户访问了攻击者构造的URL
- 服务器端不会有二次认证
- 被害者是不知情的

3.CSRF原理(引用大佬的一幅图)

(1):用户浏览并登录正常网站A
(2):用户通过认证，服务器向用户发送一个cookie值
(3):用户在没有登出A网站的情况下，访问攻击者所伪造的危险网站B(cookie值存在的时候访问网站B)
(4):网站B要求访问第三方站点网站A，发出一个请求
(5):由于此时用户仍登录着网站A(也就是说浏览器中还存放着网站A给用户的cookie)，因此浏览器会携带着之前网站A发送给自己的cookie值去访问网站A
(6):由于浏览器携带者用户正确的cookie值访问网站A，网站A会通过用户的访问请求，此时服务器仍旧认为是用户在访问网站A,实际上是攻击者B伪造用户A的身份在访问网站A，因此攻击者就可以执行用户所可以执行的相关操作了



4.CSRF例子(图示)



(1):用户Tom打开浏览器，访问受信任网站affectedBank.com，输入用户名和密码请求登录网站affectedBank.com

(2):在用户Tom信息通过验证后，网站affectedBank.com产生Cookie信息并返回给浏览器，此时用户登录网站affectedBank.com成功，可以正常发送请求到网站affectedBank.com

(3):用户Tom未退出网站affectedBank.com之前，在同一浏览器中，打开一个网页访问网站Sns.com(这个网站是攻击者Jerry提前构造好的恶意网站)

(4):网站Sns.com接收到用户Tom请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点affectedBank.com

(5):浏览器在接收到这些攻击性代码后，根据网站Sns.com的请求，在用户Tom不知情的情况下携带Cookie信息，向网站affectedBank.com发出请求。网站affectedBank.com并不知道该请求其实是由攻击者Jerry构造的恶意网站Sns.com发起的，所以会根据用户Tom的Cookie信息以Tom的权限处理该请求，导致来自网站Sns.com的恶意代码被执行。

三、CSRF与XSS组合拳

1.XSS和CSRF结合使用的原因

单纯的一个XSS的攻击效果是很小的，以至于很多厂商对于单独的XSS漏洞都是不认领的，单纯的一个CSRF的攻击要求又相对比较高，需要用户去点击攻击者构造的URL，因此CSRF的攻击也是相对来说比较难成功。但是，如果将XSS和CSRF结合使用，那么攻击效果就会呈几何倍的上升。

2.XSS结合CSRF攻击方法

存储型XSS和CSRF结合使用(原理与上图相差不大)
以下加粗的内容与CSRF有差异
(1):**首先，攻击者向网站A(存储型服务器)上传一个具有重定向到网站B功能的JS脚本**
(2):用户用户浏览并登录正常网站A
(3):用户通过认证，服务器向用户发送一个cookie值
(4):**与此同时触发了存储型XSS，用户跳转到攻击者所伪造的危险网站B**
(5):网站B要求访问第三方站点网站A，发出一个请求
(6):由于此时用户仍登录着网站A(也就是说浏览器中还存放着网站A给用户的cookie)，因此浏览器会携带着之前网站A发送给自己的cookie值去访问网站A
(7):由于浏览器携带者用户正确的cookie值访问网站A，网站A会通过用户的访问请求，此时服务器仍旧认为是用户在访问网站A,实际上是攻击者B伪造用户A的身份在访问网站A，因此攻击者就可以执行用户所可以执行的相关操作了

XSS结合CSRF与单独的CSRF最大的区别就是当用户从网站A到网站B的时候方式有所不同，XSS结合CSRF是利用XSS主动重定向到网站B，而单独的CSRF只能是通过用户有意或无意的点击来访问网站B

四、CSRF的防御方法

CSRF防护的一个重点是要对“用户凭证”进行校验处理，通过这种机制可以对用户的请求是合法进行判断，判断是不是跨站攻击的行为。因为“用户凭证”是Cookie中存储的，所以防护机制的处理对像也是Cookie的数据，我们要在防护的数据中加入签名校验，并对数据进行生命周期时间管理，就是数据过期管理。

Lapis框架是一种基于Moonscript语言开发的WEB框架，框架中有一段针对CSRF(Cross—Site Request Forgery)的防护代码， 是一种围绕时间戳和签名验证的CSRF防护设计，后来Lapis的作者Leafo老师还更新了CSRF的处理代码。

跨站攻击的本质是， 攻击者拿着你的“身份凭证”，冒充你进行的相关攻击行为。

为了防止CSRF的发生，创建Token处理机制，Token数据结构与时间、加密签名直接相关， 这么设计的的目的如上所说，是给“身份凭证”加上时间生存周期管理和签名校验管理，如果的凭证被人拿到了， 要先判断Token中的“签名”与时间戳是否都有效，再进行正常的业务处理， 这样通过对非法数据的校验过滤，来降低CSRF攻击的成功率。

1.验证请求的Referer值

验证请求的Referer值，如果Referer是以自己的网络开头的域名，则说明该请求来自网站自己，是合法的。如果Referer是其他网站域名或空白，就有可能是CSRF攻击，那么服务器就应该拒绝请求，但是此方法存在被绕过的可能。攻击者可以使用Burp抓包来修改Referer值来欺骗服务器。

2.再数据包中添加不可伪造的token值

CSRF之所以能够成功，是因为攻击者可以伪造用户的请求，由此可知，抵御CSRF攻击的关键在于:在请求中加入攻击者不能伪造的信息。例如可以在HTTP请求中以参数的形式加入一个随机产生的token，并在服务器端验证token，如果请求中没有token或者token值不正确，则认为该请求可能是CSRF攻击从而拒绝该请求。

使用token值来防御CSRF的安全性明显高于使用Referer值

五、token与Referer介绍

1.对Referer的验证，从什么角度去做，如果做，怎么杜绝问题?
对header中的referer的验证，一个是空referer，一个是referer过滤或者检测不完善。为了杜绝这种问题，在验证的白名单里，正则表达式应该写的完善。

2.针对token的测试应该注意哪些方面，会对token的哪方面做测试?
(1):针对token的攻击，一是对他本身的攻击，重放测试一次性、分析加密规则、校验方式是否准确等，二是结合信息泄露漏洞对它的获取，结合着发起组合攻击。
(2):信息泄露有可能是缓存、日志、git，也有可能是利用跨站存在信息泄露
(3):很多跳转登录都依赖token，有一个跳转漏洞加反射型跨站就可以组合成登录劫持了
(4):另外也可以结合着其他业务来描述token的安全性及设计不好怎么被绕过

六、签名与时间戳处理流程

1.Token的构成

为了防止CSRF攻击，Token要求不能重复，需要含有时间戳信息、签名信息

下面的图描述了一个token的数据构成

Token的数据结构。

```
-------------------------------------------------------------------------------|
                  msg              |     separator     |         signature
     |-----------| key     |   timestamp   |         .         |
Base64(sha256(msg))       |-----------
```

token由三部分组成:

(1):**消息[msg]**：而msg本身也有两部分组成：一部分：随机字符串，过期时间戳

(2):**分割符[separator]**：用于分隔msg部分与加密后生成的signature签名部分，这里用的是“.“

(3):**签名[signature]**:signature。signature签名，是对“msg消息”用特定算法进行加密后的串

```
token = base64(msg)格式化..base64(sha256("密锁", msg))
```

Token由被Base64的msg编码串+先256加密msg再进行Base64编码，两个串的内容结合。

2.Token的加密

首先，是按照合适的加密方法对数据进行加密。这里我们通用的就使用了sha-256散列算法，然后进行BASE64的格式转换。然后，我们需要在token串中隐含过期时间的设定，这种机制要保证，每条与服务器交互的Token有过期时间控制，一旦token过期服务器不处理请求。

3.Token的验证

当用户向服务提出访问请求时，产生Token再提交给服务器的时候，服务器需要判断token的有效性(是否过期，签名有效)，一旦传向服务器的请求中的Token异常，就可以判定是可疑行为不做处理，返回异常提示。

4.Token的校验

(1):Token解包

先把接收到的token，进行分解，以“.”为分隔符，分为msg部分+signature签名部分。

(2):比对签名

对msg部分的base64码反向decode_base64(msg)解码，再对解码后的msg明文，进行同样的encode_base64(sha256(msg))签名串转换处理。如果密锁相同，判断加密后的数据和客户端传过来的token.signature的部分是否一致。如果一致，说明这个token是有效的。

(3):判断时间过期

如果签名有效的,取出msg中的timestamp(时间戳)字段数据，与当前系统时间进行比较，如果过期时间小于当前时间，那这个token是过期的，需要重新的取得token。

七、CSRF防护实现流程

文字版本的防护原理上面讲了，下面我们将整个防护流程分解成函数实现， 直接通过代码的形式来看实现，其实比看文字描述更简单。

Lua代码如下：

```
local gen_token = function(key, expires)
   --做成一个过期时间戳。
   if expires == nil then
    expires = os.time() + 60 + 60 * 8
   end
   --对msg部分进行base64编码。
   local msg = encode_base64(
    json.encode({
        key = key,
        expires = expires
     }))
    --进行sha256哈希。
    local signature = encode_base64(hmac_sha256('testkey', msg))
    --拼接成一条token。
    return msg .. "." ..signature
end
local  val_token = function(key,token)
    --对输入数据的判空操作
      if not (token) then
     return nil, 'mssing csrf token'
    end
    --对token的msg部分，signature签名部分进行拆分。
    local msg, sig = token:match("^(.*)%.(.*)$")
    if not (msg) then
        return nil, "malformed csrf token"
    end
    sig = encoding.decode_base64(sig)
    --对解包后msg，按照相同的加密key:"testkey"，重新进行sha256哈希，比对signature，
    --如果不一致，说明这个token中的数据有问题，无效的token。
    if not (sig == hmac_sha256('testkey', msg)) then
        return nil, "invalid csrf token(bad sig)"
```

```
    end
    --对msg进行base64解码，判断其中的key和传入的key是否一致。
    --如果不一致说明token也是无效的。
    msg =json.decode(decode_base64(msg))
    if not (msg.key == key) then
        return nil, "invalid csrf token (bad key)"
    end
    --取出msg部分的时间戳，判断是否大于当前时间，如果大于，说明token过期无效了。
    if not (not msg.expires or msg.expires > os.time()) then
        return nil, "csrf token expired"
    end
end
```

因为本文提到的 CSRF防护，是Moonscript实现的，最后翻译成Lua语言， 而用的Token编码的函数与signature签名用的加密算法，也都是基于Lua库，所以下面列出了这些常用的库的相关信息。

库一览列表:http://lua-users.org/wiki/CryptographyStuff

八、核心安全算法库

要实现上文所说的Token机制，要有库函数Bash64与sha256加密的工具包库支持。

1.SecureHashAlgorithm和SecureHashAlgorithmBW

这个工具包是支持sha256加密的，而且是纯lua方法的实现，但存在一个问题就是这两个包分别依赖lua5.2和lua5.3。

大部分老系统的运行环境是lua5.1，因为大部分的生产环境都是lua5.1，因为历史原因暂时没法改变。如果要把5.2的程序移植到5.1下运行，还需要移植一个lua5.2才独有的包，这是lua5.2升级之后才有的部件：bit32,而在lua5.3中又将这个部件去掉了,移植的动力不大，lua5.1的用户可以考虑使用其他的库。

2.Lcrypt

这个包不是纯lua的实现，底层加密用的是C语言，而且额外还有依赖另外另个工具包 libTomCrypt和libTomMath，github上有源码，所以要想让这个包正常运行需要手动make安装3个源码工程。

网站:http://www.eder.us/projects/lcrypt/

3.LuaCrypto

这个包的安装用的是luarocks，就比较简单了

```
luarocks install luacrypto
```

我们选用这个包进行加密处理。LuaCrypto其实是openssl库的前端lua调用，依赖openssl，openssl库显然会支持sha256加密，相对也比一般的第三方实现更可靠。写一个简单的加密程序:

```
local crypto = require("crypto")
local hmac = require("crypto.hmac")
local ret = hmac.digest("sha256", "abcdefg", "hmackey")
print(ret)
```

ret的返回结果是，如下这个字符串。

```
704d25d116a700656bfa5a6a7b0f462efdc7df828cdbafa6fbf8b39a12e83f24
```

我们需要改造一下代码，在调用digest的时候指定输出的形式是raw二进制数据形式，然后在编码成base64的数据形式。

```
local ret = hmac.digest("sha256", "abcdefg", "hmackey",rawequal)
print(ret)
```

这时候的输出结果是：

```
cEOlORanAGVr+lpqew9GLv3H34KM26+m+/izmhLoPyQ=
lua-base64
```

使用的是下面的库，lua库就是这样，有很多功能程序有很多的实现，并且很多非官方的第三方实现。

参考网站:https://github.com/toastdriven/lua-base64

关于库的部分就介绍这些，可以找一个可运行lua的运行环境实践这个处理过程。