Burp it out

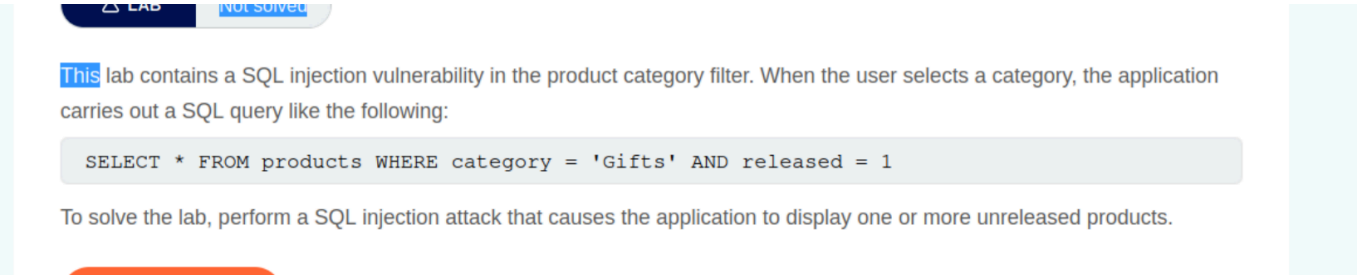
Amaan Abbas

My write-up from the summer of 2025, where I delved into web exploitation using Burp Suite Academy and other resources. Many notes are from the Burp Suite website. I have added the techniques I used for thinking while solving some of them.

I used this resource extensively while trying to identify bugs during my internship, and it proved helpful in finding those bugs.

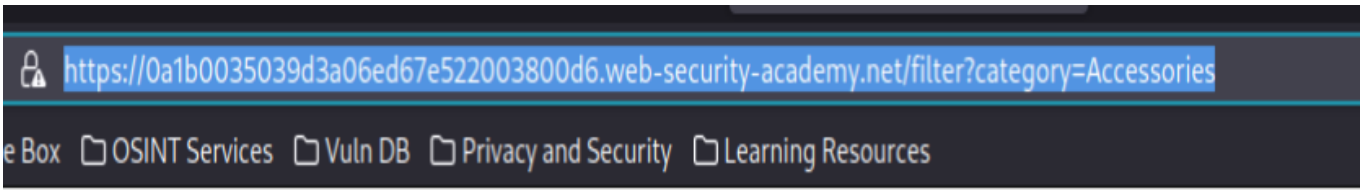
Also other resources links have been added...

SQL Injection

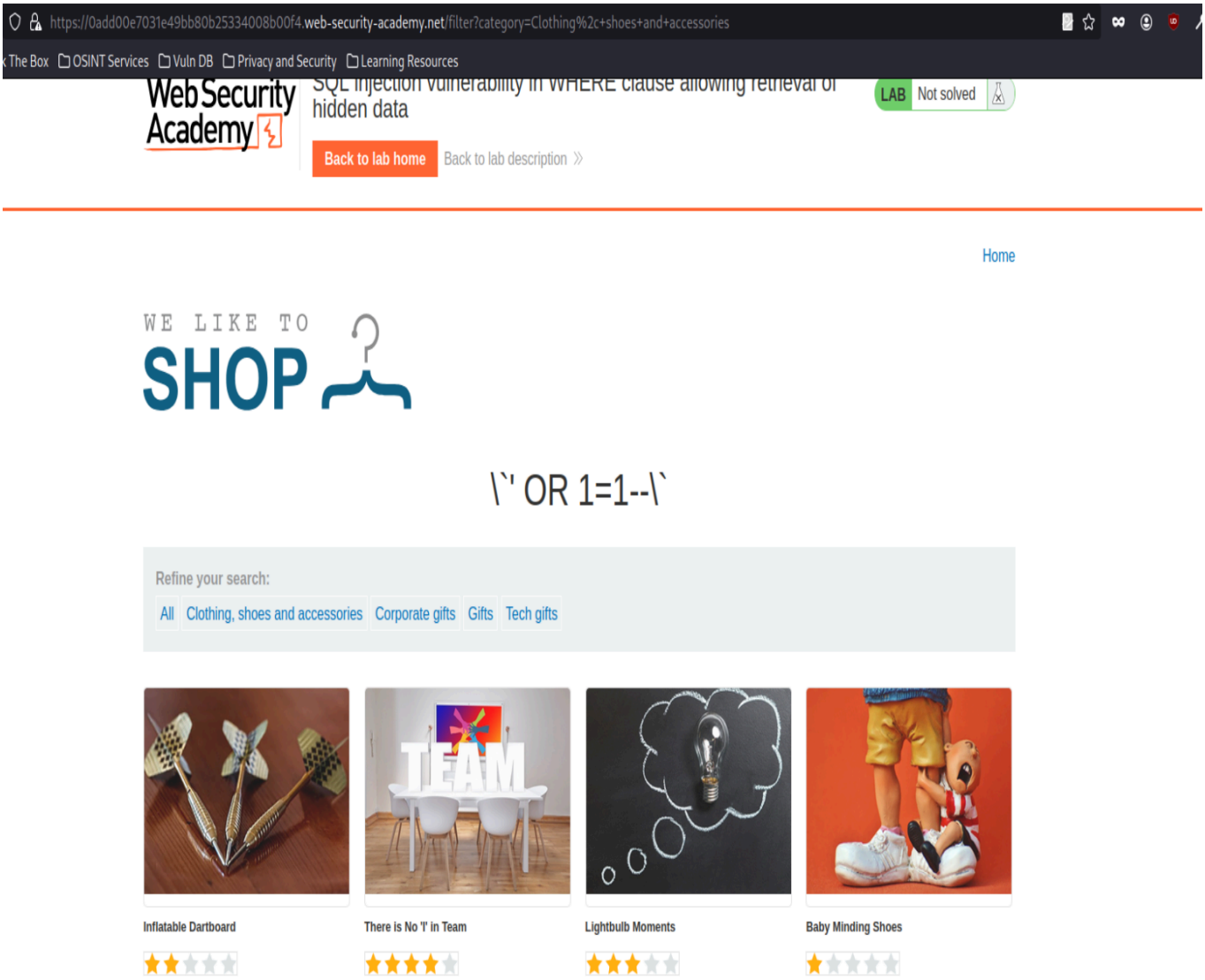


Since we are given SELECT * from ..... where = "gifts"...

We can assume if we put the category = `'+OR+1=1--` --- a classic sql injection we might get what we want



on the request we can see catergory=""..maybe if we change it?..



and we do...pretty easy all i had to do is intercept the request make a change and send it..on the burp suite academy it should show complete now..

Now for the next task we are given:

Imagine an application that lets users log in with a username and password. If a user submits the username `wiener` and the password `bluecheese`, the application checks the credentials by performing the following SQL query:

```
SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
```

If the query returns the details of a user, then the login is successful. Otherwise, it is rejected.

In this case, an attacker can log in as any user without the need for a password. They can do this using the SQL comment sequence `--` to remove the password check from the `WHERE` clause of the query. For example, submitting the username `administrator'--` and a blank password results in the following query:

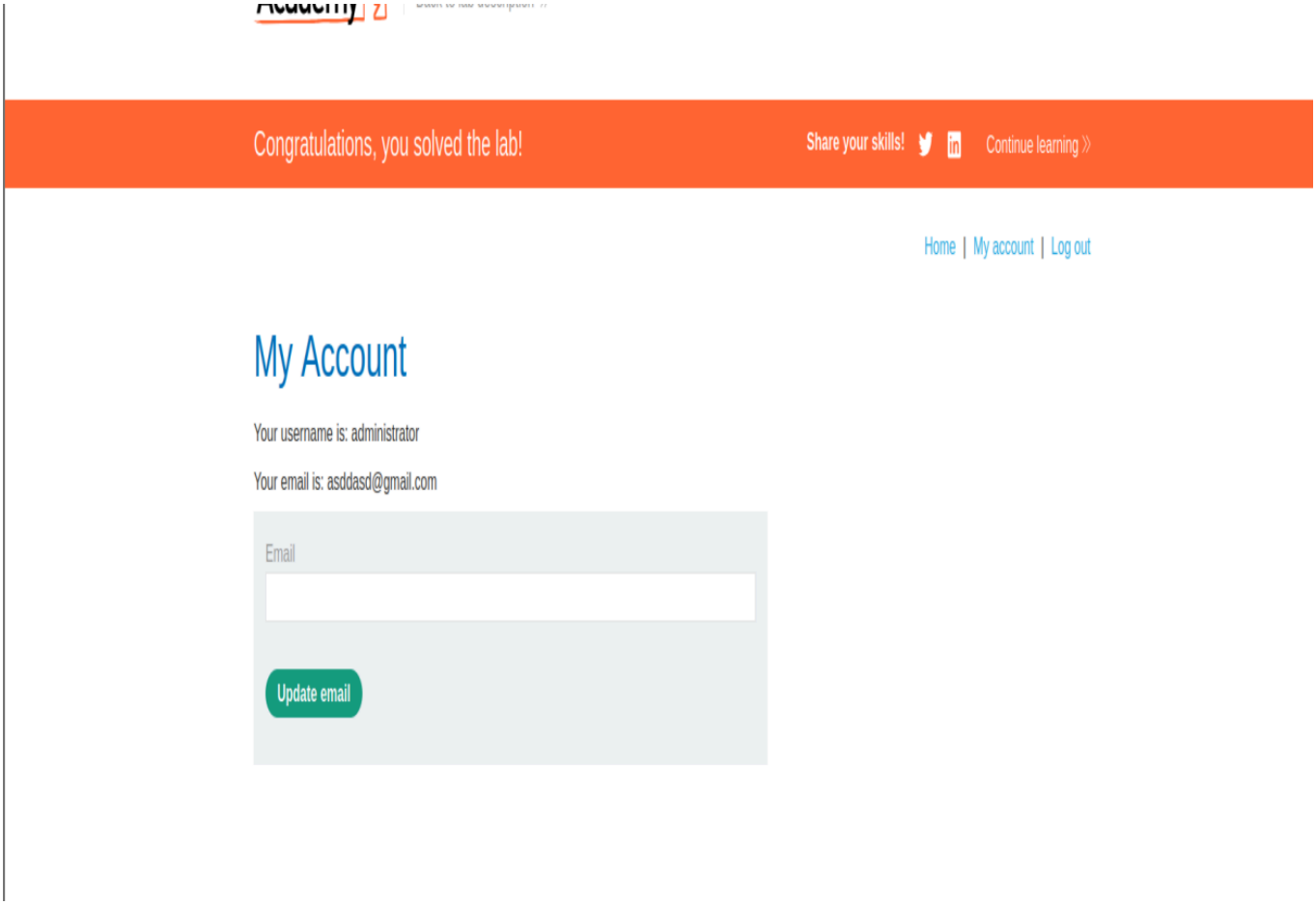
```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

This query returns the user whose `username` is `administrator` and successfully logs the attacker in as that user.

Lets try it now:

First time i tried with
'administrator'--' and '' it should error..

after that i learned if we just do it as adminstrator'--' it ignores anything after that and i also did it using burp intercept and the needful changes and got



yay!!

Now imma try an oracle database injection:

On Oracle databases, every `SELECT` statement must specify a table to select `FROM`. If your `UNION SELECT` attack does not query from a table, you will still need to include the `FROM` keyword followed by a valid table name.

There is a built-in table on Oracle called `dual` which you can use for this purpose. For example: `UNION SELECT 'abc' FROM dual`

For more information, see our SQL injection cheat sheet.

Concept: `UNION SELECT`

- `UNION` lets you combine results from **multiple SELECT queries**.
- You can use it to **inject your own data** into the results of the original query.

🧪 Original query (example):

Suppose a vulnerable query looks like:

```
SELECT name, password FROM users WHERE name = '$input'
```

If `$input = 'anything' UNION SELECT 'abc','def' FROM dual--`, then it becomes:

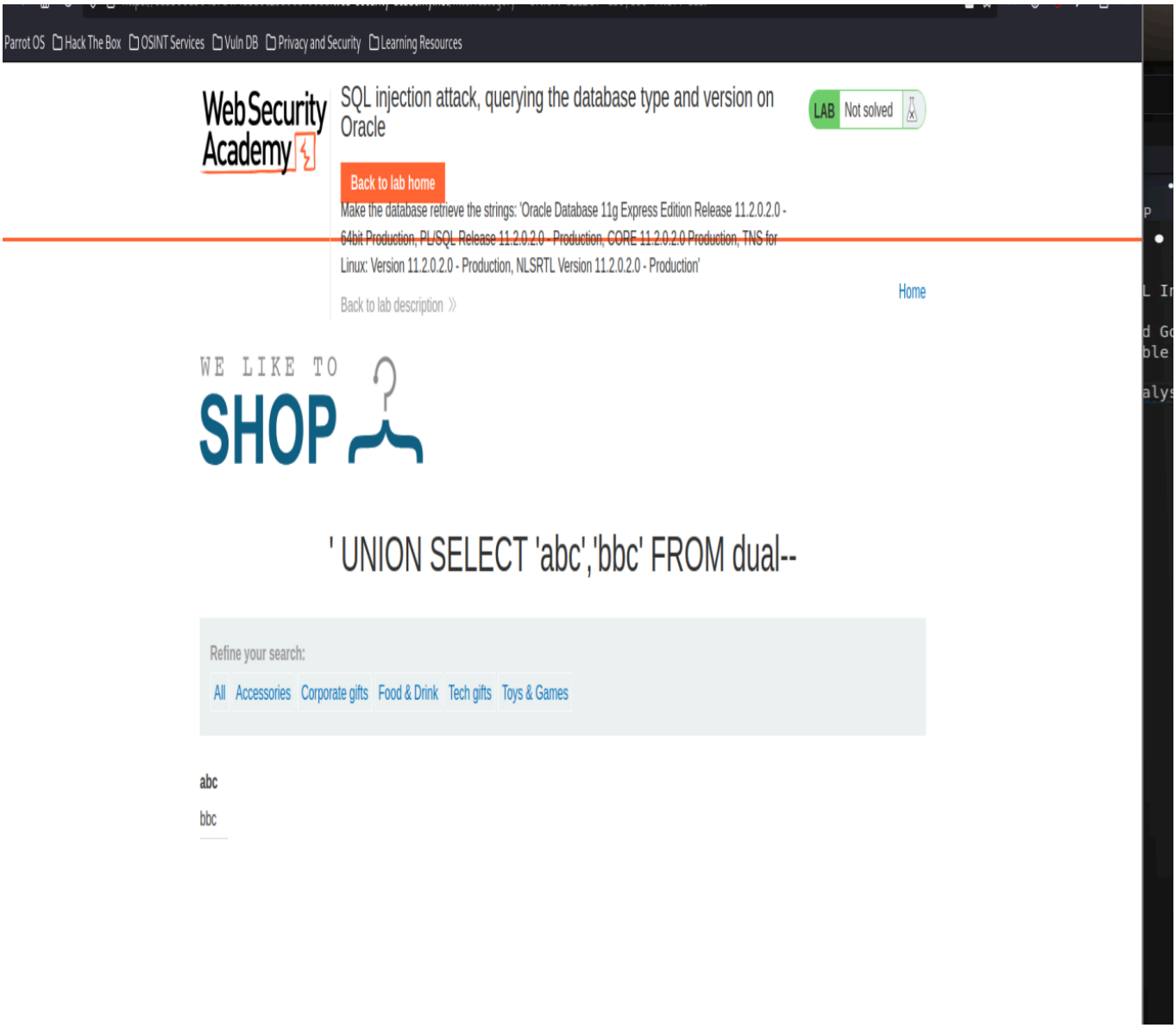
```
SELECT name, password FROM users WHERE name = 'anything' UNION SELECT 'abc', 'def' FROM dual --'
```

> `dual` is a **dummy table** in Oracle and MySQL (some versions) used to return one row.

📜 Final Output:

- First result: may be empty if `name = 'anything'` doesn't exist.
- Second result: returns `abc`, `def`.

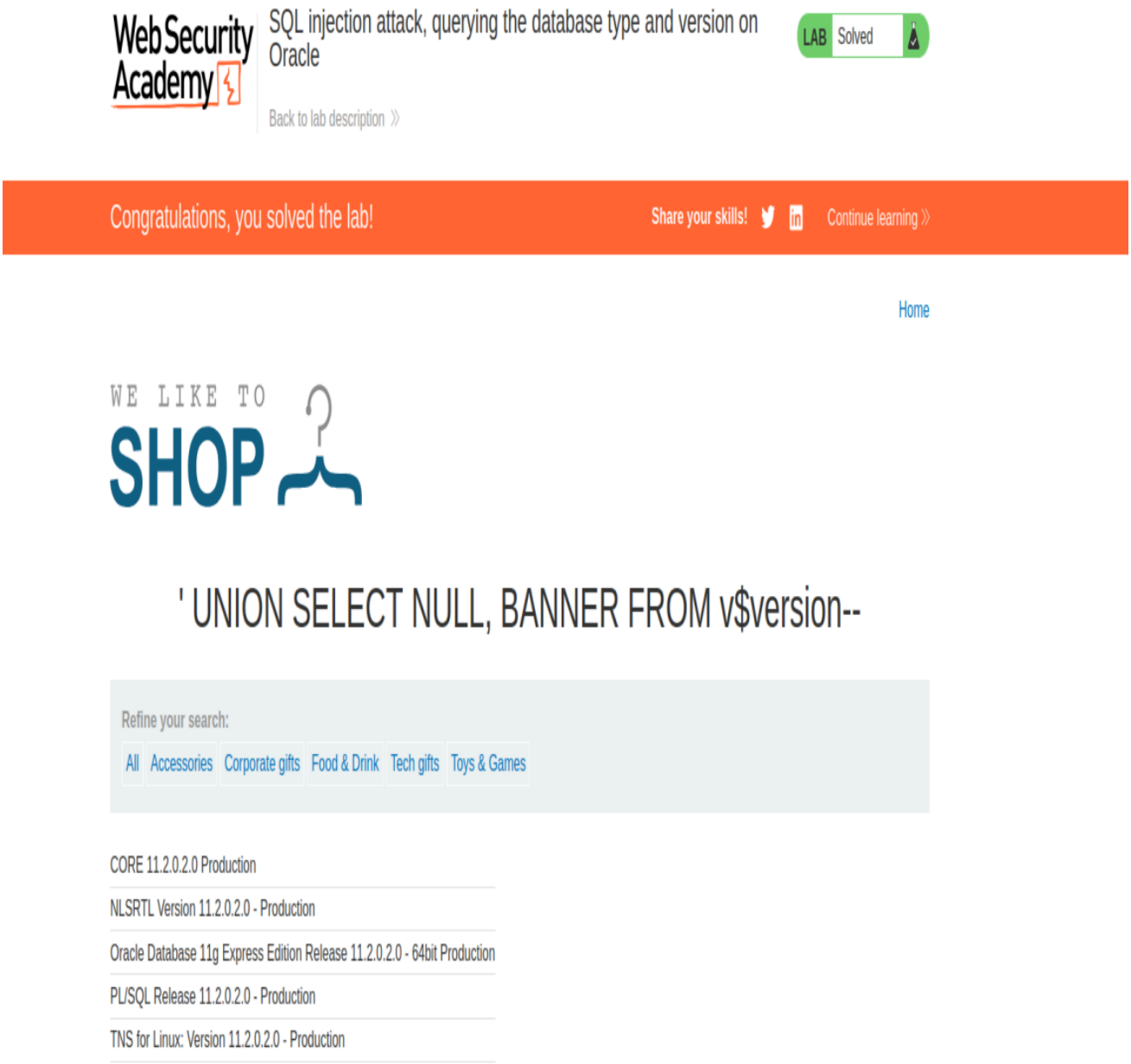
You just **inject a fake row** into the output.



Now we know err there are 2 couloumns on the table:)
hence we can:

- `UNION` = same # of columns.
- `NULL` = dummy to match structure.
- `BANNER` = real data you want.
  UNION SELECT NULL, BANNER FROM v$version--

and we get the answer



Blind SQL injection

In this section, we describe techniques for finding and exploiting blind SQL injection vulnerabilities.

What is blind SQL injection?

Blind SQL injection occurs when an application is vulnerable to SQL injection, but its HTTP responses do not contain the results of the relevant SQL query or the details of any database errors.

Many techniques such as `UNION` attacks are not effective with blind SQL injection vulnerabilities. This is because they rely on being able to see the results of the injected query

within the application's responses. It is still possible to exploit blind SQL injection to access unauthorized data, but different techniques must be used.

Exploiting blind SQL injection by triggering conditional responses

Consider an application that uses tracking cookies to gather analytics about usage. Requests to the application include a cookie header like this:

```
Cookie: TrackingId=u5YD3PapBcR4lN3e7Tj4
```

When a request containing a `TrackingId` cookie is processed, the application uses a SQL query to determine whether this is a known user:

```
SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'u5YD3PapBcR4lN3e7Tj4'
```

This query is vulnerable to SQL injection, but the results from the query are not returned to the user. However, the application does behave differently depending on whether the query returns any data. If you submit a recognized `TrackingId`, the query returns data and you receive a "Welcome back" message in the response.

This behavior is enough to be able to exploit the blind SQL injection vulnerability. You can retrieve information by triggering different responses conditionally, depending on an injected condition.

To understand how this exploit works, suppose that two requests are sent containing the following `TrackingId` cookie values in turn:

```
…xyz' AND '1'='1 …xyz' AND '1'='2
```

- The first of these values causes the query to return results, because the injected `AND '1'='1` condition is true. As a result, the "Welcome back" message is displayed.
- The second value causes the query to not return any results, because the injected condition is false. The "Welcome back" message is not displayed.

This allows us to determine the answer to any single injected condition, and extract data one piece at a time.

For example, suppose there is a table called `Users` with the columns `Username` and `Password`, and a user called `Administrator`. You can determine the password for this user by sending a series of inputs to test the password one character at a time.

To do this, start with the following input:

```
xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) > 'm
```

This returns the "Welcome back" message, indicating that the injected condition is true, and so the first character of the password is greater than `m`.

Next, we send the following input:

```
xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) > 't
```

This does not return the "Welcome back" message, indicating that the injected condition is false, and so the first character of the password is not greater than `t`.

Eventually, we send the following input, which returns the "Welcome back" message, thereby confirming that the first character of the password is `s`:

```
xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) = 's
```

We can continue this process to systematically determine the full password for the `Administrator` user.

Ok for this question you had to first query a length parameter and get each character out it was pretty tricky i have attached a video below if you want to take a look and this is a ss of me trying it

the &a& thing can be changed for each character of the password where we brute force and try to get the answer..so we get each letter when the length is diffrent and move on..i have made a simple if else for help:

if letter found:
go next character:
try again:
till all 20 met:

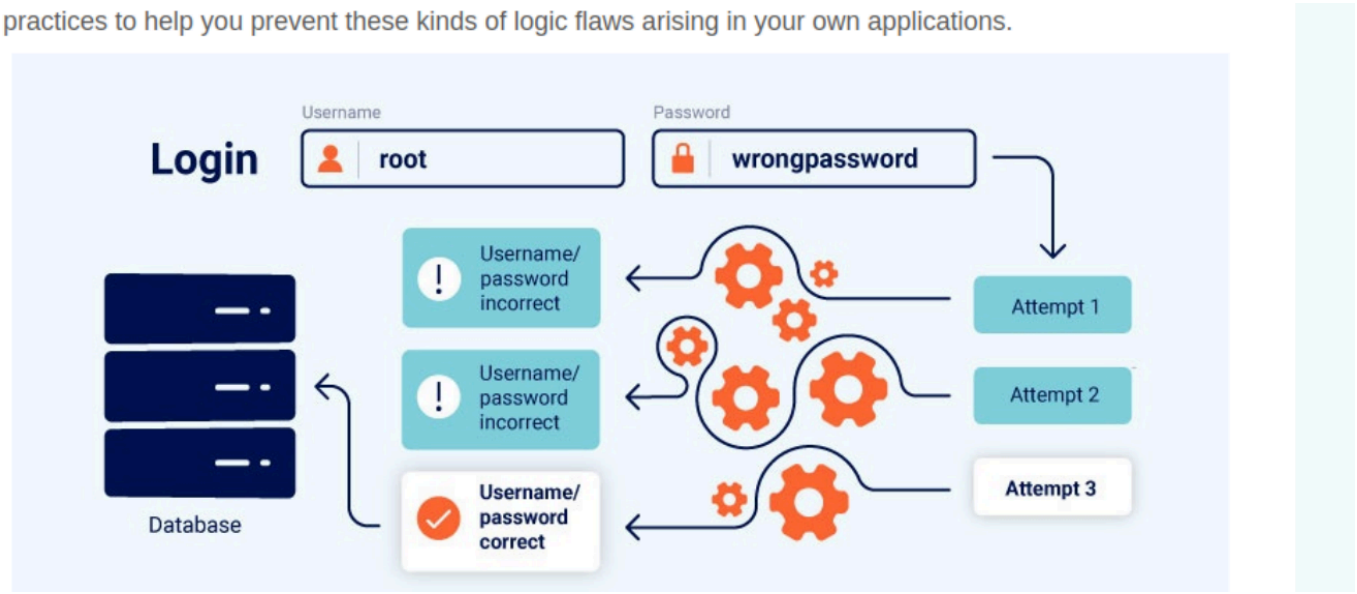
tricky question lol!
https://www.youtube.com/watch?v=xrFK_YQRxPg

Prevention methods:
https://portswigger.net/web-security/sql-injection#how-to-prevent-sql-injection

---

Ok we done with SQL now
In this section, we'll introduce the concept of business logic vulnerabilities and explain how they can arise due to flawed assumptions about user behavior. We'll discuss the potential impact of

logic flaws and teach you how they can be exploited. You can also practice what you've learned using our interactive labs, which are based on real bugs that we've encountered in the wild. Finally, we'll provide some general best practices to help you prevent these kinds of logic flaws arising in your own applications.[from Potswigger]



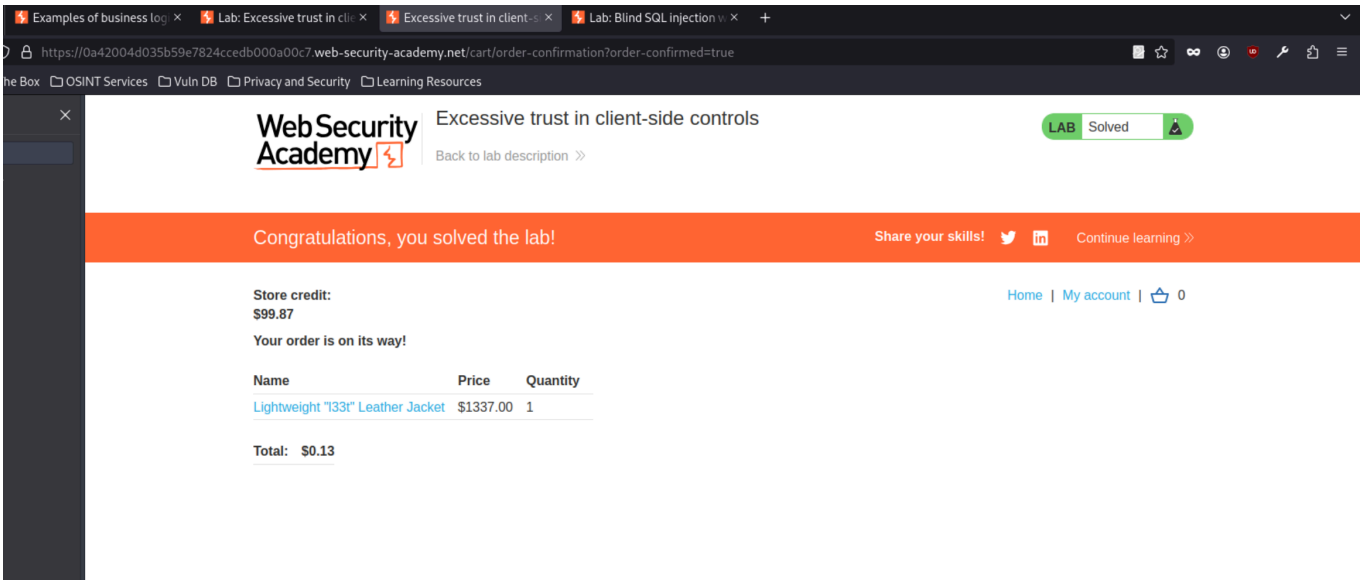
Logic-based vulnerabilities can be extremely diverse and are often unique to the application and its specific functionality. Identifying them often requires a certain amount of human knowledge, such as an understanding of the business domain or what goals an attacker might have in a given context. This makes them difficult to detect using automated vulnerability scanners. As a result, logic flaws are a great target for bug bounty hunters and manual testers in general.

Business logic vulnerabilities often arise because the design and development teams make flawed assumptions about how users will interact with the application. These bad assumptions can lead to inadequate validation of user input. For example, if the developers assume that users will pass data exclusively via a web browser, the application may rely entirely on weak client-side controls to validate input. These are easily bypassed by an attacker using an intercepting proxy.

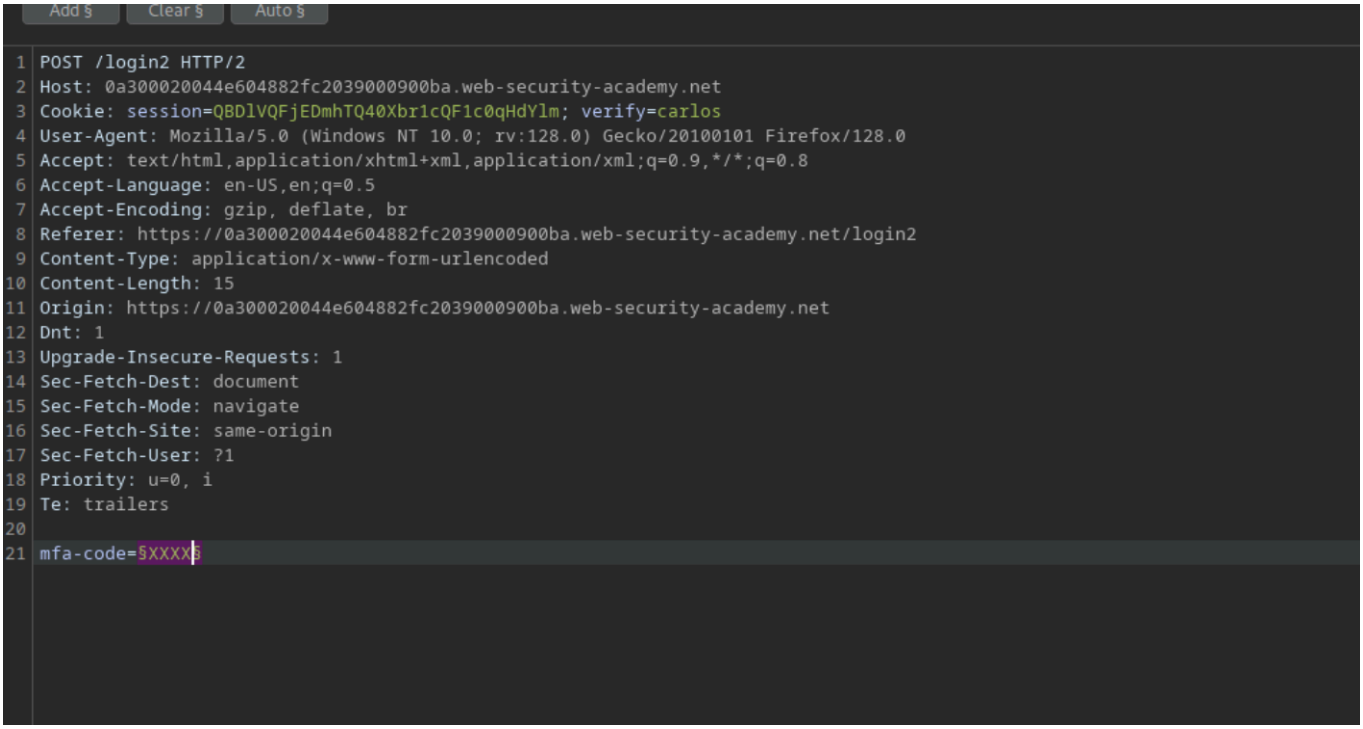
Ultimately, this means that when an attacker deviates from the expected user behavior, the application fails to take appropriate steps to prevent this and, subsequently, fails to handle the situation safely.

For this question below all i had to was intercept the cart where price is given and make changes to make it cheap

The reason its called business logic is...they dont think customers is gona do this but rather just focus on buying the product lol..



Another one they had was a 2FA one...so u have a account with a password...what u do is u intercept the mfa request as show below:

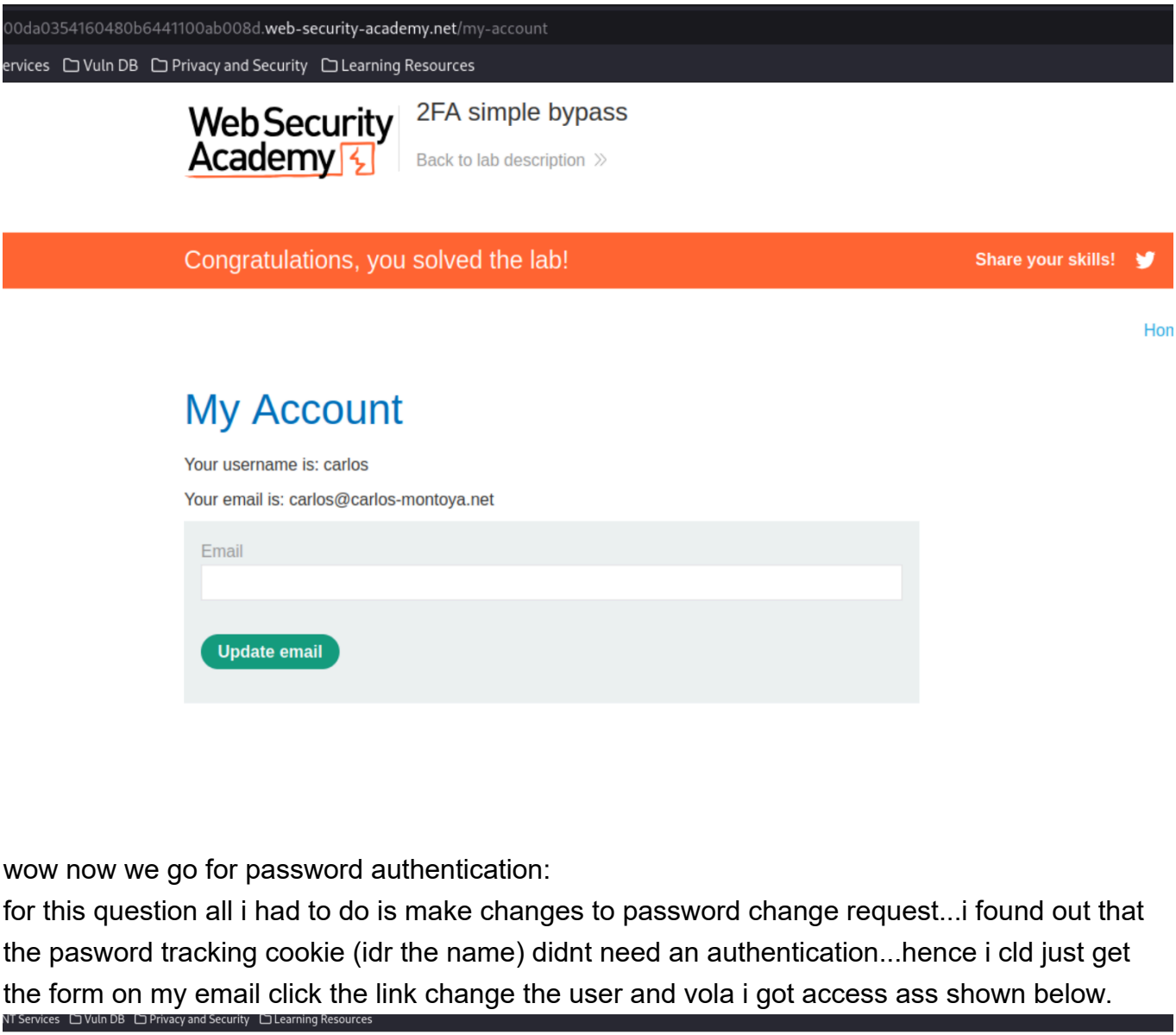


what we can do is change the verify parameter to any username...then keep spamming the mfa code to get a 4 digit which is correct....

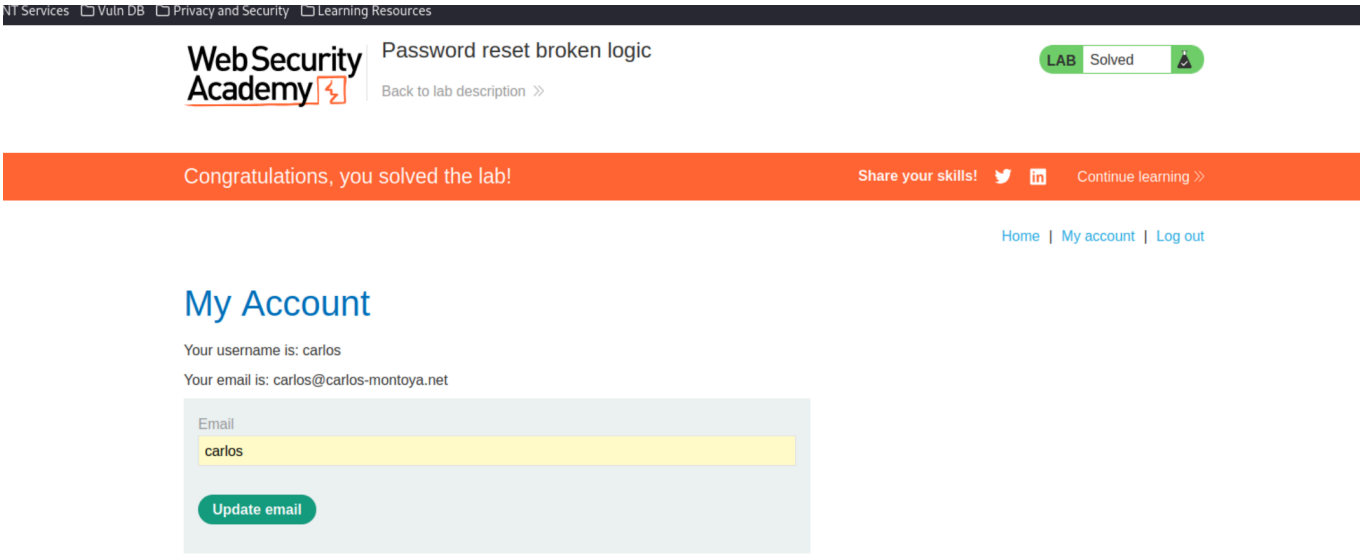
so what we did:
1)able to change the user name to who we want to verify without knowing thier password
2)spam mfa 4 digit code since it doesn't have limit of request before stopping.

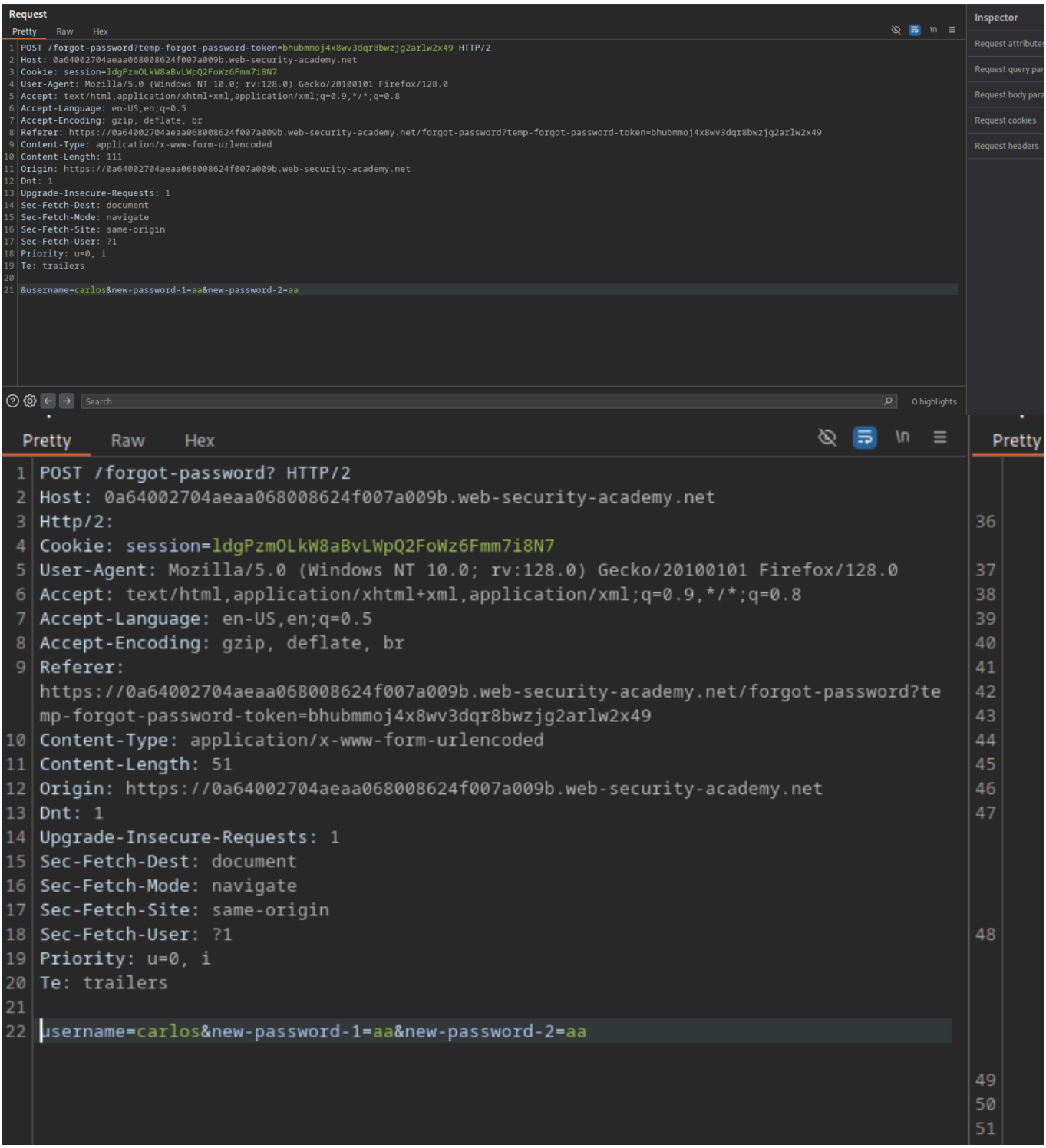
THE ABOVE WAS a harder one.

below is ther easier one...where i was given an account to bypass mfa all i had to do is redirect the webpage



wow now we go for password authentication:
for this question all i had to do is make changes to password change request...i found out that the pasword tracking cookie (idr the name) didnt need an authentication...hence i cld just get the form on my email click the link change the user and vola i got access ass shown below.





Ok for this question it was a bit tricky:

what i was given:

a)a username and account, an exploit page, a hint for X-Forward

I then realized after a lot of trial and errors that there when the forgot password was clicked we could send a x forward to the website we want which here was the exploit server we were given

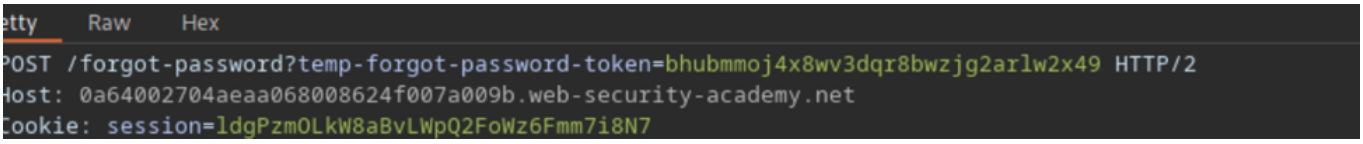
and hence send a malicious request as shown below

Displaying all emails @exploit-0ae100b303a41f7f800f9365017b0061.exploit-server.net and all subdomains

Sent	To	From	Subject	Body	
2025-07-07 13:50:02 +0000	wiener@exploit-0ae100b303a41f7f800f9365017b0061.exploit-server.net	no-reply@0a540082035b1f2a8098947700a0008b.web-security-academy.net	Account recovery	Hello! Please follow the link below to reset your password. https://exploit-0ae100b303a41f7f800f9365017b0061.exploit-server.net/forgot-password?temp-forgot-password-token=gynmusv9qgma6slvi77m3o085ylkmozx Thanks, Support team	View raw

now i could send this same crafted request to carlos to get the reset password-token..

on my "exploit website where i put the X-Forward so it comes to me" which btw is (X-Forwarded-Host: exploit-0a4700e6032142978153 47f101660062.exploit-server.net)....i get the token succesfuly since carlos intiated from his "email domain"....so using that i craft a packet and change password..



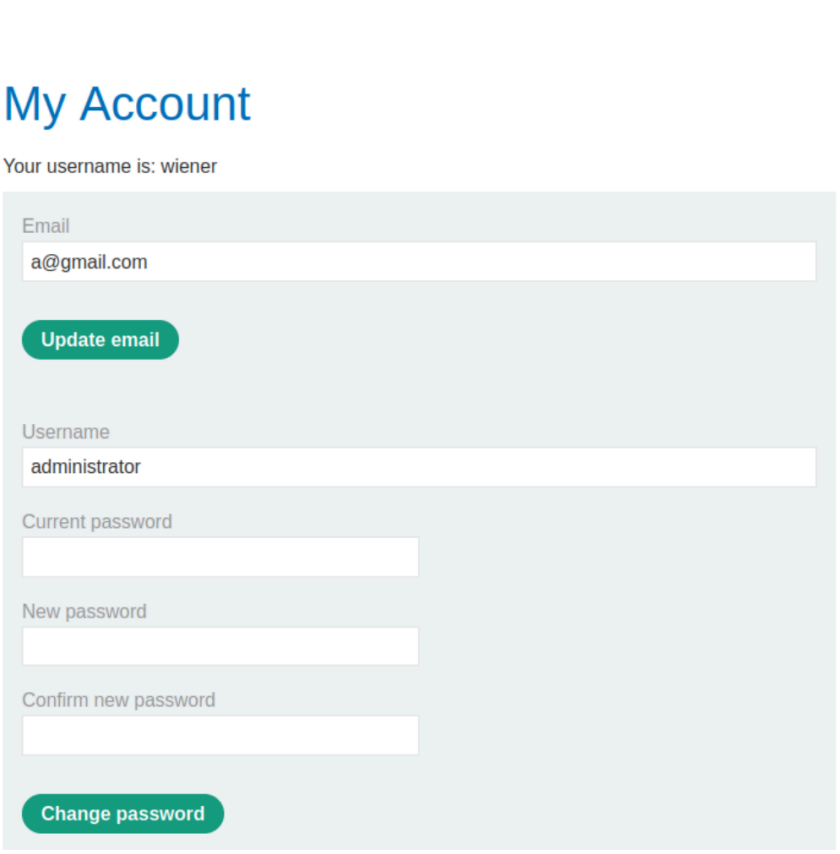
```
etty   Raw   Hex
POST /forgot-password?temp-forgot-password-token=bhubmmoj4x8wv3dqr8bwzjg2arlw2x49 HTTP/2
Host: 0a64002704aeaa068008624f007a009b.web-security-academy.net
Cookie: session=ldgPzmOLkW8aBvLWpQ2FoWz6Fmm7i8N7
```

as u see the for-pasword-token was available to me when carlos clicked on my crafted request that directs his info to me..

cool question

---

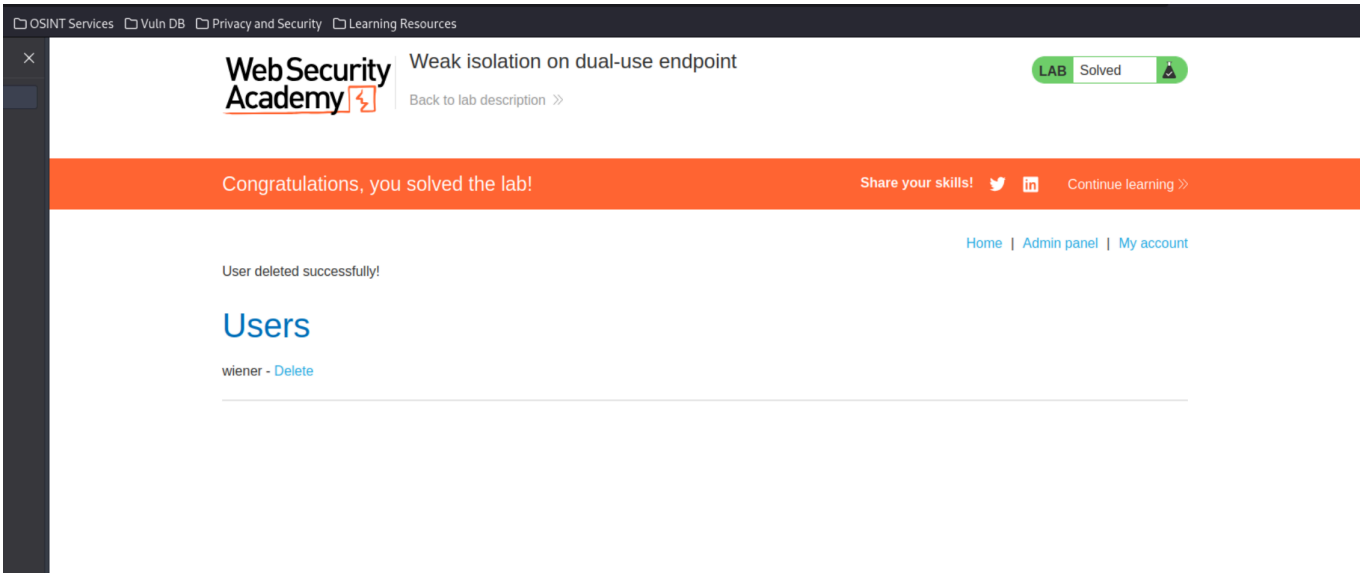
for this next question all i had to do is choose a username i wanted to change password from on the change request...for me it was administrator... as shown below



intercept the packet and then...remove current password parameter..and send...tada we got the adminstrator

```
Pretty  Raw  Hex
1  POST /my-account/change-password HTTP/2
2  Host: 0afa0074037ea662835c06cc005400ca.web-security-academy.net
3  Cookie: session=6bETwJMt96lRm2F6FgH3fZ36VPp21i2Y
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0
5  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate, br
8  Referer:
   https://0afa0074037ea662835c06cc005400ca.web-security-academy.net/my-account?id=wien
   er
9  Content-Type: application/x-www-form-urlencoded
10 Content-Length: 94
11 Origin: https://0afa0074037ea662835c06cc005400ca.web-security-academy.net
12 Dnt: 1
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18 Priority: u=0, i
19 Te: trailers
20
21 csrf=u2OTivLBCesIMbxdw2LNdWbTYO1B0HWV&username=administrator&new-password-1=a&
   new-password-2=a
```

best part?..we get to delete any account as shown below and win the lab...



Dangerous scenarios can occur when user-controllable input is encrypted and the resulting ciphertext is then made available to the user in some way. This kind of input is sometimes

known as an "encryption oracle". An attacker can use this input to encrypt arbitrary data using the correct algorithm and asymmetric key.

This becomes dangerous when there are other user-controllable inputs in the application that expect data encrypted with the same algorithm. In this case, an attacker could potentially use the encryption oracle to generate valid, encrypted input and then pass it into other sensitive functions.

This issue can be compounded if there is another user-controllable input on the site that provides the reverse function. This would enable the attacker to decrypt other data to identify the expected structure. This saves them some of the work involved in creating their malicious data but is not necessarily required to craft a successful exploit.

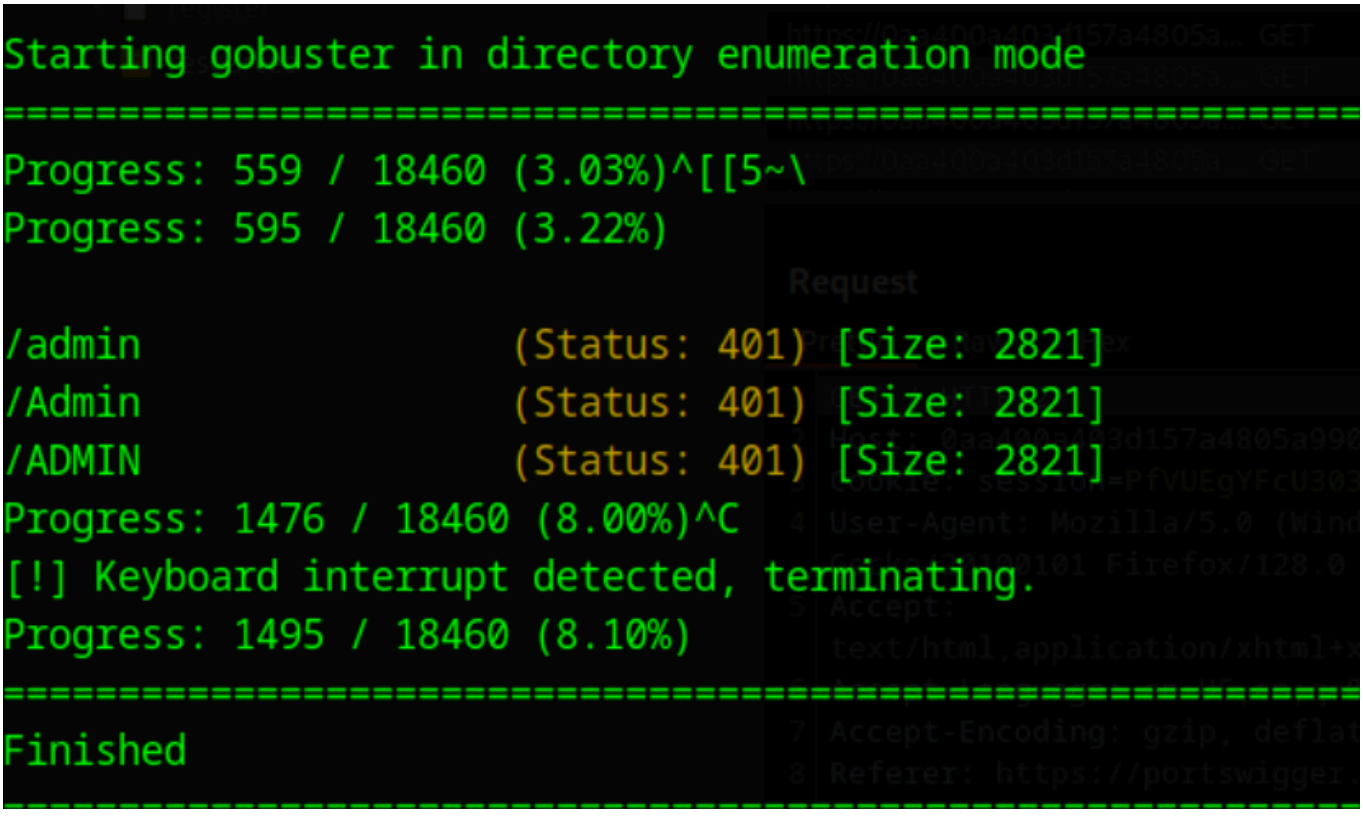
The severity of an encryption oracle depends on what functionality also uses the same algorithm as the oracle. [portswiggert]

the lab for this question was too tricky i skipped it sadly however here is the vid for the walkthrough

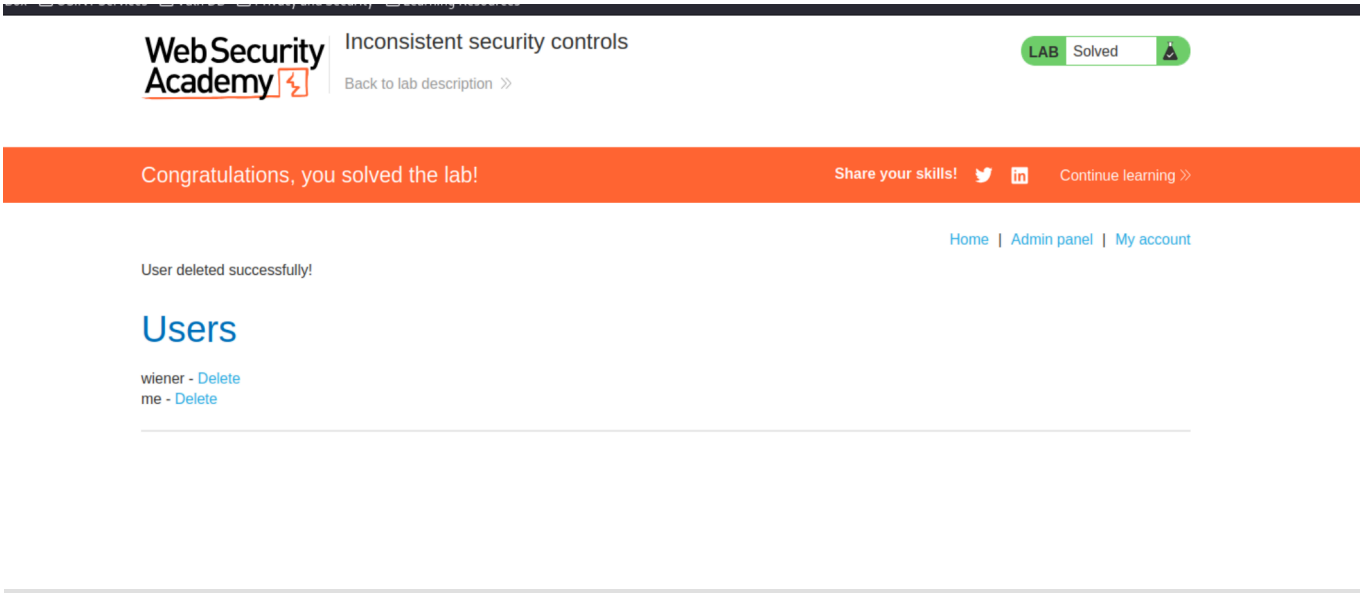
https://www.youtube.com/watch?v=62spVp-GVPI&t=545s

---

ok so we got a pretty simple but yet confusing question...they tell us to use burp professional...but honestlty just use gobuster on the domain and u will find a /admin directory...this part confused me since i had to use dirbuster



the /admin directory tells you it needs a login from @dontwannacry.com domain ....from this we can actually make a normal account and go to the "change email place" and make a random account with @dontwannacry.com and when we acces /admin we get admin privlages..this is super rare irl icl lol...however just access this and go to the accounts and delete and u finish it..



This question was an expert level and omg I enjoyed it heavy...so it used utf-7 encoding issues.. below is one mail address parser..

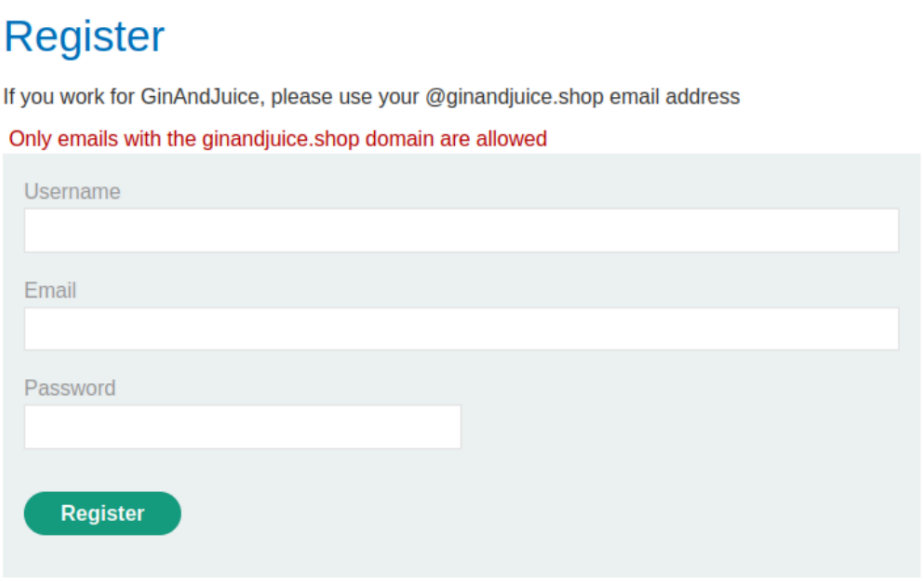
Some websites parse email addresses to extract the domain and determine which organization the email owner belongs to. While this process may initially seem straightforward, it is actually very complex, even for valid RFC-compliant addresses.

Discrepancies in how email addresses are parsed can undermine this logic. These discrepancies arise when different parts of the application handle email addresses differently.

An attacker can exploit these discrepancies using encoding techniques to disguise parts of the email address. This enables the attacker to create email addresses that pass initial validation checks but are interpreted differently by the server's parsing logic.

The main impact of email address parser discrepancies is unauthorized access. Attackers can register accounts using seemingly valid email addresses from restricted domains. This enables them to gain access to sensitive areas of the application, such as admin panels or restricted user functions...[portswigger website]...

ok since i knew i have to target the email thing for this challenge...i was given a register form



as you can see above we need a ginandjuice.shop domain....

I first tried a utf-8 encoding scheme to try getting access but that didn't work... i don't have the picture but like above the red sentence showed for this case that the login was blocked due to security reasons....hence i assumed utf-8 was blocked...and gave a try for utf-7....tbh i dint know how utf-7 encoding cld be created like how the encoding scheme is....hence i just chatgpted and looked online then i learned that for this question what we are trying to do is have our email domain and have the rest of it commented someway....i have the email written below...

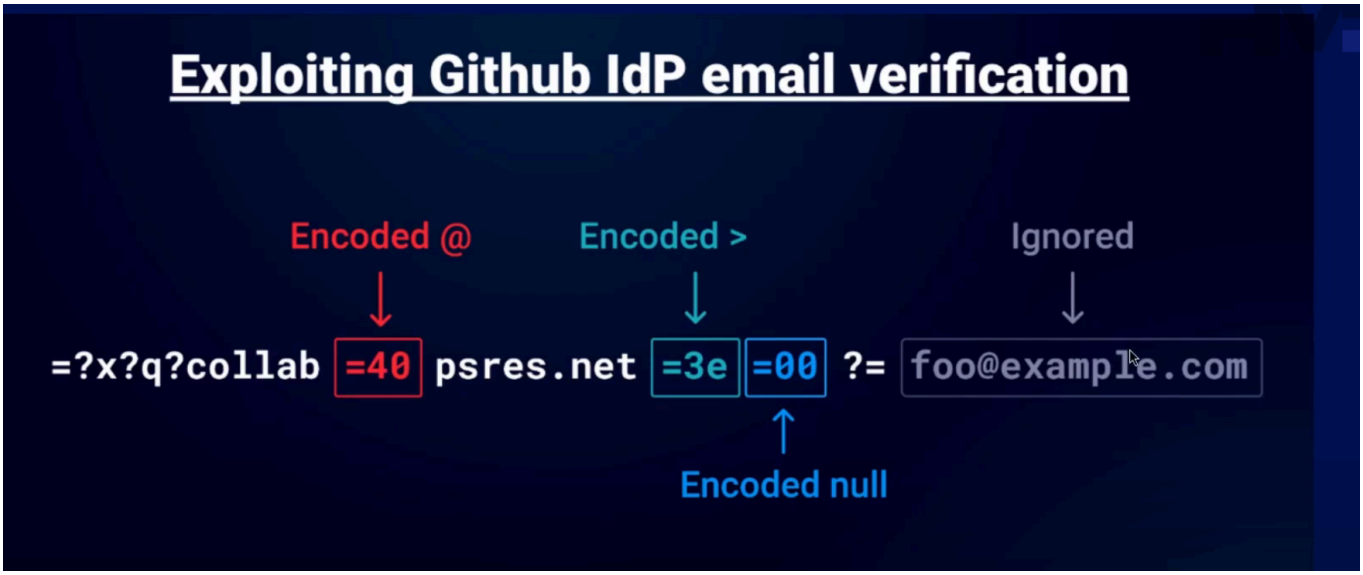
```
=?utf-7?q?attacker&AEA-exploit-0a8000650380dd348181ec3801aa0026.exploit-
server.net&ACA-?=@ginandjuice.shop
```

looks scary lol but it aint less break it down

Part	Meaning
=?utf-7?q?	Start of MIME encoded-word using UTF-7 + quoted-printable encoding
`attacker`	Just plain text, part of the local name
`&AEA-`	Encoded `@` character in UTF-7
`exploit-0a8000650380dd...net`	Your exploit-server domain, encoded normally

Part	Meaning
`&ACA-`	Encoded **space** character in UTF-7 — breaks the domain visually/logically
`?=`	End of MIME encoded-word(commented rest of stuff)
`@ginandjuice.shop`	Added plain text to **trick domain filter** — NOT used by the mail system

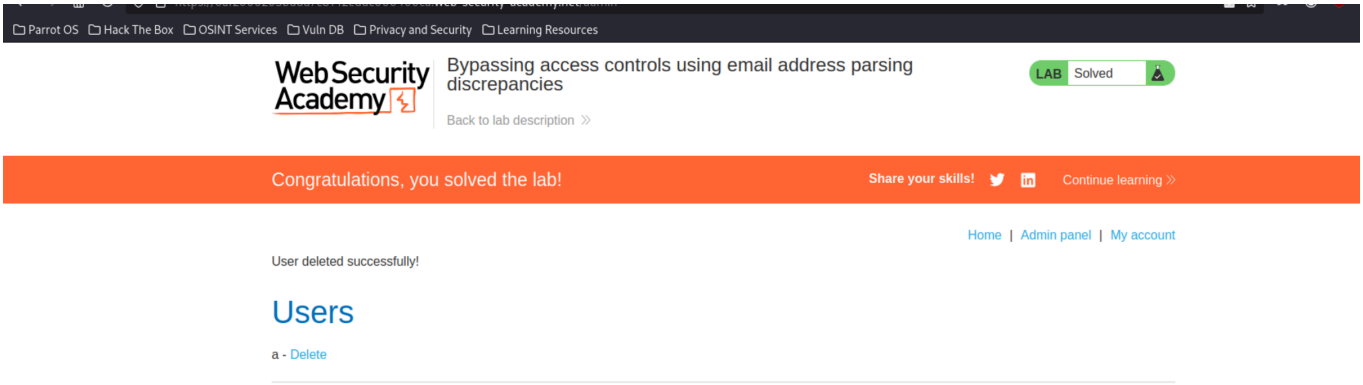
pretty cool ye??...here is another image from the guy who found it



Also i have added his blog below

https://portswigger.net/research/splitting-the-email-atom

Anyways



all i had to do is click the link in my email make an account then i cld delete a user and solve the lab

Hurray!!

What is access control?

Access control is the application of constraints on who or what is authorized to perform actions or access resources. In the context of web applications, access control is dependent on authentication and session management:

- **Authentication** confirms that the user is who they say they are.
- **Session management** identifies which subsequent HTTP requests are being made by that same user.
- **Access control** determines whether the user is allowed to carry out the action that they are attempting to perform.

Broken access controls are common and often present a critical security vulnerability. Design and management of access controls is a complex and dynamic problem that applies business, organizational, and legal constraints to a technical implementation. Access control design decisions have to be made by humans so the potential for errors is high.[portswigger]

https://portswigger.net/web-security/access-control

Broken access control resulting from platform misconfiguration

Some applications enforce access controls at the platform layer. they do this by restricting access to specific URLs and HTTP methods based on the user's role. For example, an application might configure a rule as follows:

```
DENY: POST, /admin/deleteUser, managers
```

This rule denies access to the `POST` method on the URL `/admin/deleteUser`, for users in the managers group. Various things can go wrong in this situation, leading to access control bypasses.

Some application frameworks support various non-standard HTTP headers that can be used to override the URL in the original request, such as `X-Original-URL` and `X-Rewrite-URL`. If a website uses rigorous front-end controls to restrict access based on the URL, but the application allows the URL to be overridden via a request header, then it might be possible to bypass the access controls using a request like the following:

```
POST / HTTP/1.1 X-Original-URL: /admin/deleteUser ...
```

Parameter-based access control methods

Some applications determine the user's access rights or role at login, and then store this information in a user-controllable location. This could be:

- A hidden field.
- A cookie.
- A preset query string parameter.

The application makes access control decisions based on the submitted value. For example:

`https://insecure-website.com/login/home.jsp?admin=true https://insecure-website.com/login/home.jsp?role=1`

Referer-based access control

Some websites base access controls on the `Referer` header submitted in the HTTP request. The `Referer` header can be added to requests by browsers to indicate which page initiated a request.

For example, an application robustly enforces access control over the main administrative page at `/admin`, but for sub-pages such as `/admin/deleteUser` only inspects the `Referer` header. If the `Referer` header contains the main `/admin` URL, then the request is allowed.

In this case, the `Referer` header can be fully controlled by an attacker. This means that they can forge direct requests to sensitive sub-pages by supplying the required `Referer` header, and gain unauthorized access.

Horizontal privilege escalation

Horizontal privilege escalation occurs if a user is able to gain access to resources belonging to another user, instead of their own resources of that type. For example, if an employee can access the records of other employees as well as their own, then this is horizontal privilege escalation.

Horizontal privilege escalation attacks may use similar types of exploit methods to vertical privilege escalation. For example, a user might access their own account page using the following URL:

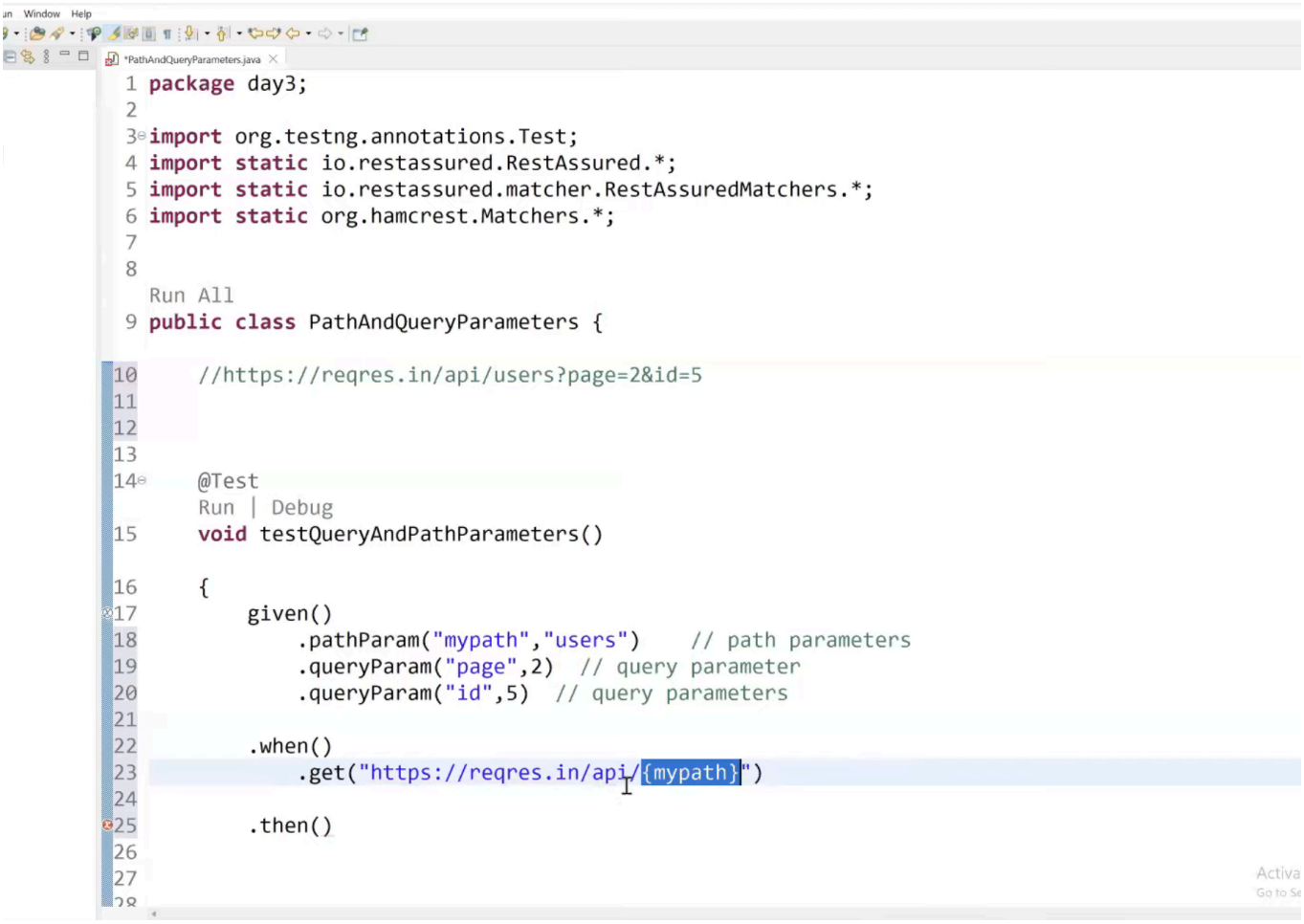
`https://insecure-website.com/myaccount?id=123`

If an attacker modifies the `id` parameter value to that of another user, they might gain access to another user's account page, and the associated data and functions.

API path vs Query Parameters:

Aspect	Query String	API Parameters (Path Params)
📍 **Location**	After `?` in the URL	Inside the URL path
🧱 **Structure**	`?key=value&key2=value2`	`/resource/{id}` or `/user/123`
📘 **Example**	`/search?term=hello&page=2`	`/user/123` or `/posts/42/comments`
🔁 **Use Case**	Filters, sorting, pagination, optional stuff	Identifying a specific resource (mandatory)
🛠️ **Editable?**	Easy to modify manually	Fixed in endpoint unless API changes
📜 **Read by**	`req.query` in Express.js	`req.params` in Express.js

![[]]



Information disclosure in error message

I was recently doing an internship on pen testing and came accross this.....also burp got labs on it aswell which is great

A vulnerability where the application unintentionally reveals sensitive data, such as usernames, internal IPs, software versions, source code, or credentials—often due to misconfigurations, verbose errors, or debug info.

examples of this cold be robot.txt but for our lab we found a special one on internal error could come adn the version is leaked(the version according to burp was vulnerable)...

All i had to do was intercept the traffic when reloading the url

```
https://0aba009e041a04558012fdc800c600cd.web-security-academy.net/product?
productId=1

over here i was able to change productid to "1"---string

this created an error
```

sometimes **Source code disclosure via backup files** this happens...for a question i had too...

```
go to robots.txt see that /backup allowed
go there and see everything till i found the password stored in it
```

sometimes websites have stuff hidden in their git where we can browse to `/.git` to reveal the lab's Git version control data.....

Command	Description
`git log`	Shows commit history
`git show`	Shows content of the latest commit
`git show <commit-hash>`	Shows content of a specific commit
`git status`	Shows the working directory status
`git branch`	Lists branches
`git checkout <branch>`	Switches to a different branch
`git diff`	Shows unstaged changes
`git diff <commit1> <commit2>`	Compares two commits

Command	Description
`git blame <file>`	Shows who changed each line of a file and when
`git log --oneline`	Condensed log view (one line per commit)
`git log -p`	Shows diff of each commit
`git log --name-only`	Shows files changed in each commit
`git remote -v`	Lists remote repositories
`git tag`	Lists tags (can indicate releases/versions)
`git grep <string>`	Searches for a string in the repo history
`git ls-files`	Lists all tracked files
`git rev-list --all`	Lists all commit hashes
`git cat-file -p <hash>`	Views a git object (commit, tree, blob) by hash
`git restore <file>`	Restores a file to the latest committed state
`git checkout <hash> -- <file>`	Recovers a file version from a specific commit

the above are the commands for this question i was able to do git show and see what all commits was shown then from git show "hash" I was able to find what was needed to get the password

`TRACE /admin` is an HTTP request using the `TRACE` method. Here's what it does:

Component	Explanation
`TRACE`	An HTTP method used to echo back the received request—used mainly for debugging.
`/admin`	The target path on the server (commonly an admin panel).

Why It's Important (Security Context):

- If a server responds to `TRACE`, it may reflect headers like cookies or authorization tokens.
- **Vulnerability:** This can be abused for **Cross Site Tracing (XST)** if combined with XSS. might give(`X-Custom-IP-Authorization`
-)

API recon

To start API testing, you first need to find out as much information about the API as possible, to discover its attack surface.

To begin, you should identify API endpoints. These are locations where an API receives requests about a specific resource on its server. For example, consider the following `GET` request:

```
GET /api/books HTTP/1.1 Host: example.com
```

The API endpoint for this request is `/api/books`. This results in an interaction with the API to retrieve a list of books from a library. Another API endpoint might be, for example, `/api/books/mystery`, which would retrieve a list of mystery books.

Once you have identified the endpoints, you need to determine how to interact with them. This enables you to construct valid HTTP requests to test the API. For example, you should find out information about the following:

- The input data the API processes, including both compulsory and optional parameters.
- The types of requests the API accepts, including supported HTTP methods and media formats.
- Rate limits and authentication mechanisms.[portswigger]

If you identify an endpoint for a resource, make sure to investigate the base path. For example, if you identify the resource endpoint `/api/swagger/v1/users/123`, then you should investigate the following paths:

- `/api/swagger/v1`
- `/api/swagger`
- `/api`

Burp sequencer:

Burp Sequencer analyzes how random session tokens are
It helps you find out if tokens can be predicted
If tokens are weak, you might guess others' tokens
That can let you hijack sessions or impersonate users
Sequencer itself only analyzes tokens you capture, it doesn't grab them automatically
You use Sequencer results to decide if guessing tokens is possible
Then you try to use predicted tokens to access other accounts manually or with scripts

diffrent type of api vid
https://www.youtube.com/watch?v=2mqN7ZhDsUA

Using machine-readable documentation

You can use a range of automated tools to analyze any machine-readable API documentation that you find.

You can use Burp Scanner to crawl and audit OpenAPI documentation, or any other documentation in JSON or YAML format. You can also parse OpenAPI documentation using the OpenAPI Parser BApp.

You may also be able to use a specialized tool to test the documented endpoints, such as Postman or SoapUI.

While browsing the application, look for patterns that suggest API endpoints in the URL structure, such as `/api/`. Also look out for JavaScript files. These can contain references to API endpoints that you haven't triggered directly via the web browser. Burp Scanner automatically extracts some endpoints during crawls, but for a more heavyweight extraction, use the JS Link Finder BApp. You can also manually review JavaScript files in Burp.The Param miner BApp enables you to automatically guess up to 65,536 param names per request. Param miner automatically guesses names that are relevant to the application, based on information taken from the scope

Identifying supported HTTP methods

The HTTP method specifies the action to be performed on a resource. For example:

- `GET` - Retrieves data from a resource.
- `PATCH` - Applies partial changes to a resource.
- `OPTIONS` - Retrieves information on the types of request methods that can be used on a resource.

An API endpoint may support different HTTP methods. It's therefore important to test all potential methods when you're investigating API endpoints. This may enable you to identify additional endpoint functionality, opening up more attack surface.

For example, the endpoint `/api/tasks` may support the following methods:

- `GET /api/tasks` - Retrieves a list of tasks.
- `POST /api/tasks` - Creates a new task.
- `DELETE /api/tasks/1` - Deletes a task.

Some times Changing the content type may enable you to trigger faults in the api

Once you have identified some initial API endpoints, you can use Intruder to uncover hidden endpoints. For example, consider a scenario where you have identified the following API

endpoint for updating user information:

```
PUT /api/user/update
```

To identify hidden endpoints, you could use **Burp Intruder** to find other resources with the same structure. For example, you could add a payload to the `/update` position of the path with a list of other common functions, such as `delete` and `add`.

When looking for hidden endpoints, use wordlists based on common API naming conventions and industry terms. Make sure you also include terms that are relevant to the application, based on your initial recon [burp suite academy]

Some systems contain internal APIs that aren't directly accessible from the internet. Server-side parameter pollution occurs when a website embeds user input in a server-side request to an internal API without adequate encoding. This means that an attacker may be able to manipulate or inject parameters, which may enable them to, for example:

- Override existing parameters.
- Modify the application behavior.
- Access unauthorized data

A good resource for better understanding use this article here

You can use an URL-encoded `&` character to attempt to add a second parameter to the server-side request.

For example, you could modify the query string to the following:

```
GET /userSearch?name=peter%26foo=xyz&back=/home
```

This results in the following server-side request to the internal API:

```
GET /users/search?name=peter&foo=xyz&publicProfile=true
```

Review the response for clues about how the additional parameter is parsed. For example, if the response is unchanged this may indicate that the parameter was successfully injected but ignored by the application.

Encoding to Bypass Filters

If filters block `&` or =, encode them:

- `%26` = `&`
- `%3D` = =

Example:

```
https://example.com/api/search?name=peter%26name=administrator
```

https://www.youtube.com/watch?v=eWEgUcHPle0 --- a vid abt cross site scripting

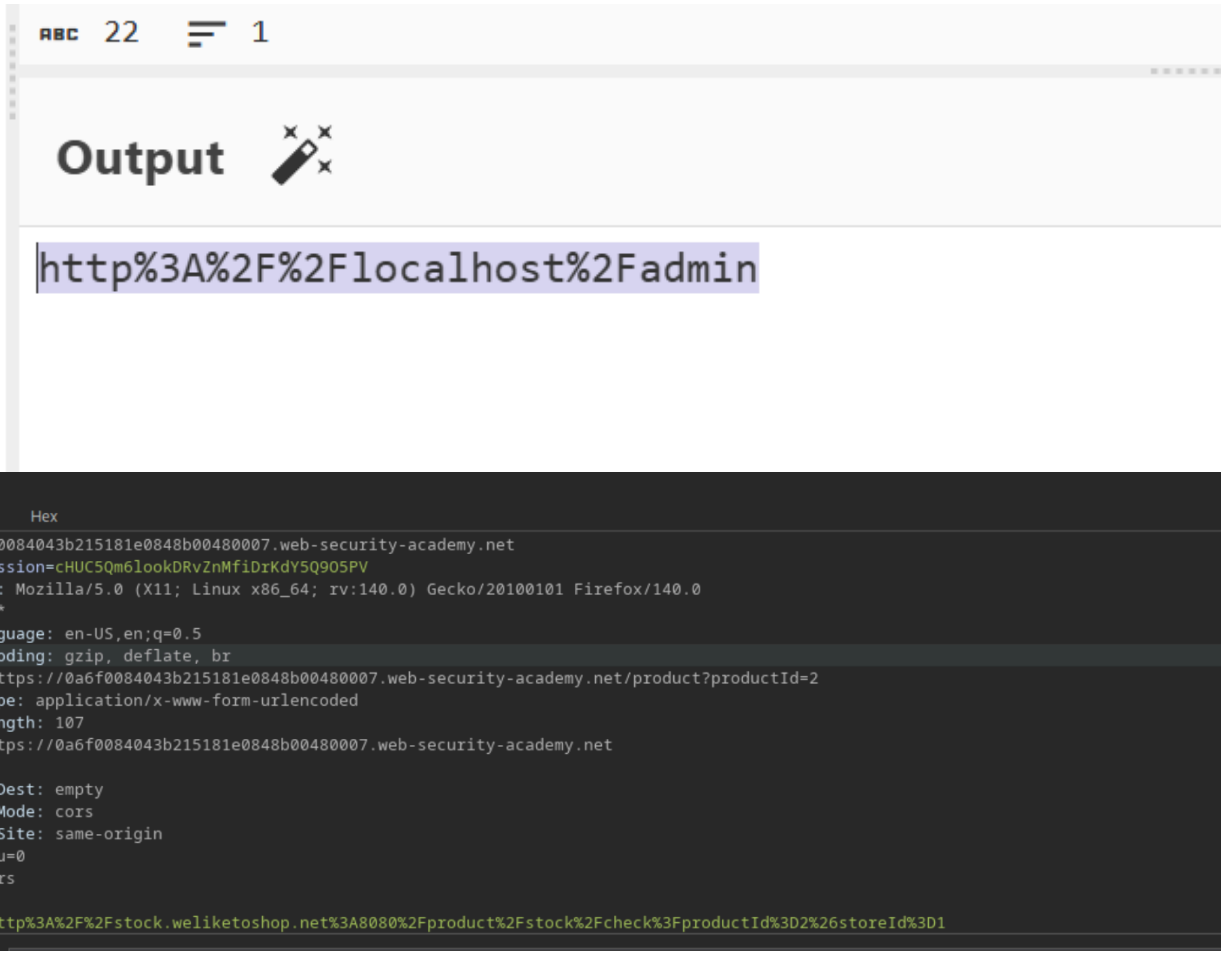
https://developer.mozilla.org/en-US/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_URL- --good documentation for url's

What is SSRF?

Server-side request forgery is a web security vulnerability that allows an attacker to cause the server-side application to make requests to an unintended location.

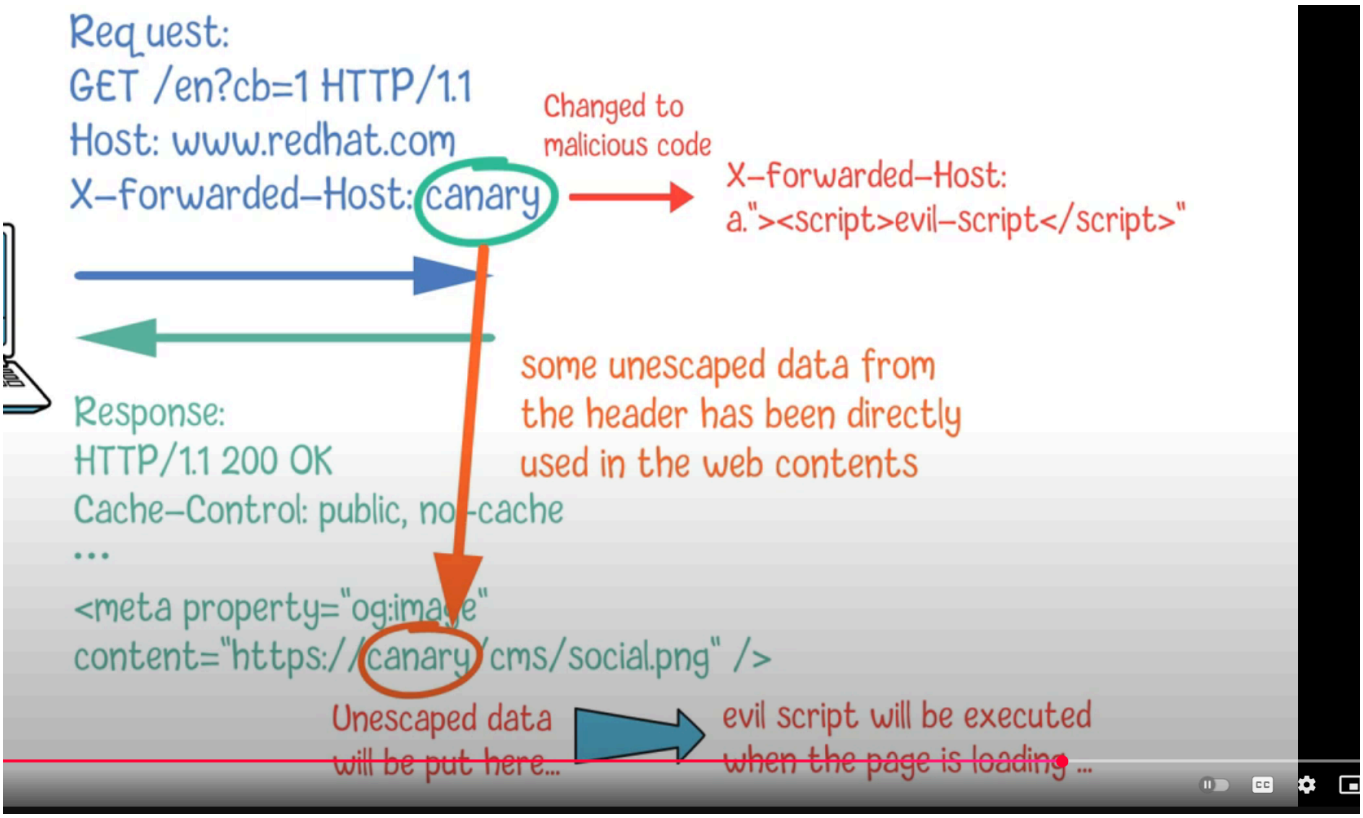
https://www.youtube.com/watch?v=ih5R_c16bKc --ssrf--- https://www.youtube.com/watch?v=voTHFdL9S2k

https://portswigger.net/web-security/api-testing/top-10-api-vulnerabilities --OWASP TOP 10 for api

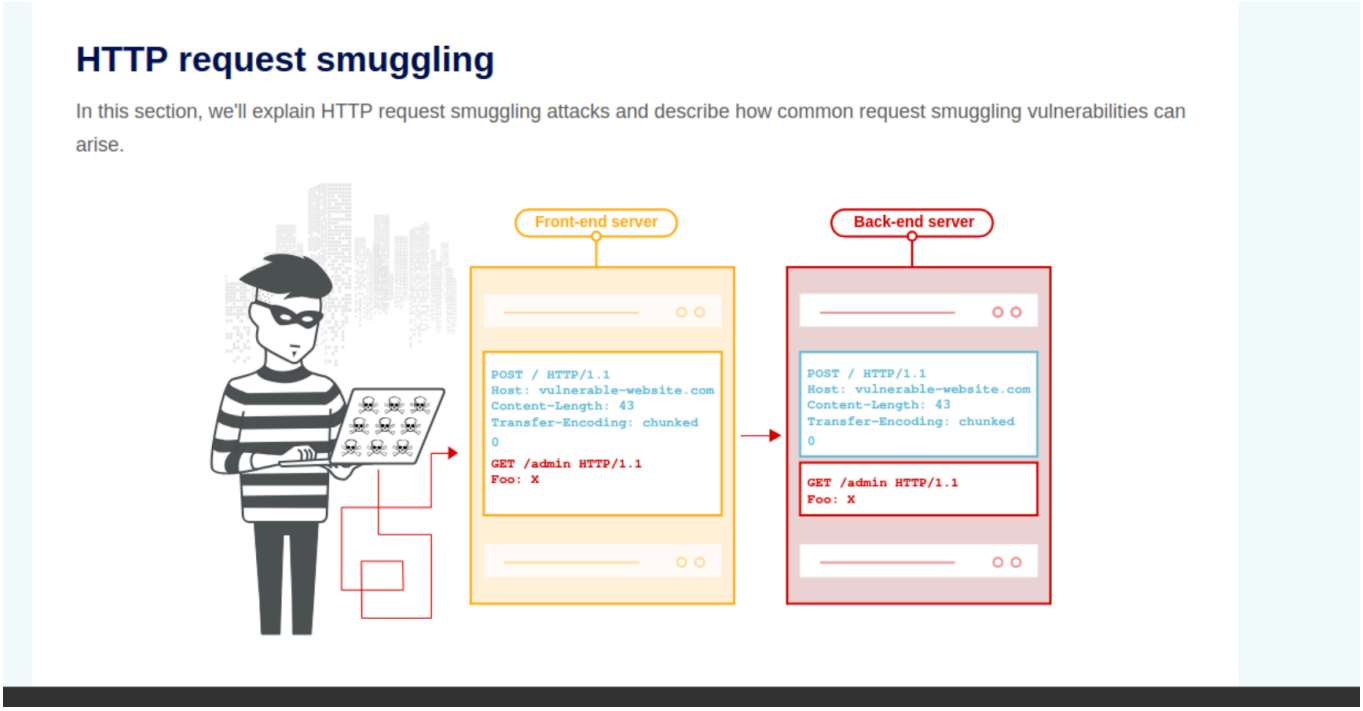


https://portswigger.net/web-security/ssrf/url-validation-bypass-cheat-sheet --- to create url bypasses

Web Cache Poisoning:



https://www.youtube.com/watch?v=mroq9eHFOIU --- Omer blackhat web caching



I will be ending this writeup with another useful resource OWASP Top 10

```
Dataview (inline field '?utf-7?q?attacker&AEA-exploit-
0a8000650380dd348181ec3801aa0026.exploit-server.net&ACA-?
=@ginandjuice.shop'): Error:
-- PARSING FAILED -----------------------------------------------
-

> 1 | ?utf-7?q?attacker&AEA-exploit-
0a8000650380dd348181ec3801aa0026.exploit-server.net&ACA-?
=@ginandjuice.shop
    | ^

Expected one of the following:

'(', 'null', boolean, date, duration, file link, list ('[1, 2,
3]'), negated field, number, object ('{ a: 1, b: 2 }'), string,
variable
```