Chad Ballay

CYBR350-342N

Week 6

08/08/2020

WebGoat – Journal pt 3 (A4-A5pt1)

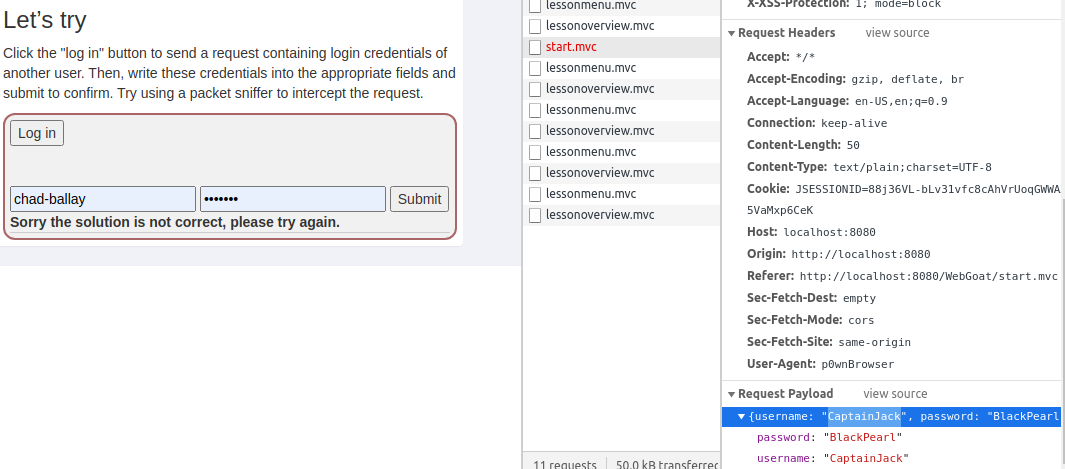
Please see previous doc for full notes.

<https://github.com/Chad-Ballay/Cybersecurity-work-class/blob/master/classes/CYBR350/week6/journal_notes.docx>

# Sensitive Data Exposure

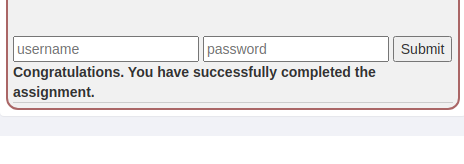
## Insecure Login

Starts off with prompting us to attempt to login and sniff the traffic.



To do so I looked at the traffic on the network layer.

Upon seeing that, I then attempted to use it and got the Congratulations text.



The big lesson I saw here is don’t trust user input and to always rotate keys and passwords. This will help enforce best practices as well as limit the danger of someone going rogue unnoticed.

# XML External Entities

XXE

I kinda understand the concept of this one. You are using XML to define the action and mechanism to give you the access you need. Kinda. The part for me that I don’t seem to grasp is the how the details work. I really am lost on the XML to put in the form. This is even after using someone else’s walk through to try and understand how it worked. (<https://trello.com/c/NNmTGWXu/4-xxe>)

curl 'https://127.0.0.1:8080/WebGoat/xxe/simple'

-H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:79.0)

Gecko/20100101 Firefox/79.0' -H 'Accept: \*/\*' -H 'Accept-Language:

en-US,en;q=0.5' --compressed -H 'Content-Type: application/xml'

-H 'X-Requested-With: XMLHttpRequest' -H 'Origin: https://127.0.0.1:8080'

-H 'Connection: keep-alive' -H 'Referer: https://127.0.0.1:8080/WebGoat/start.mvc'

-H 'Cookie: JSESSIONID=dSwqNJzuoeeS16-7B82TNitzjDpr08rhGnyntaRH'

-H 'Pragma: no-cache' -H 'Cache-Control: no-cache'

--data-raw $'<\041DOCTYPE comment [\n<\041ELEMENT text (#PCDATA)>\n

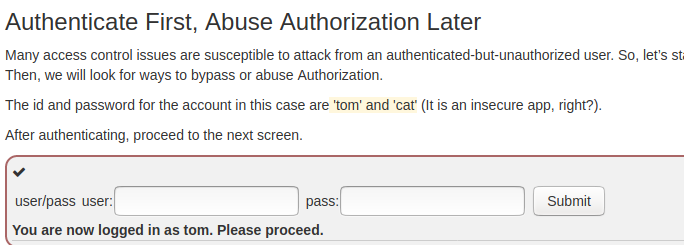
<\041ENTITY js SYSTEM "file:///">\n]>\n<comment> <text>&js;</text></comment>'

Cutting and pasting someone else’s solution isn’t helping me a this point so I’m putting a pause on continuing for this one.

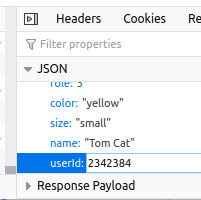
# Broken Access Control

# Insecure Direct Object References

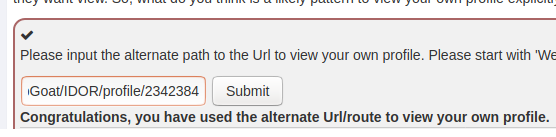
This one is common enough that I’ve inadvertently abused it on accident. At work we’ve got a portal that is used as the record of truth to see if you passed some internal training. The page to load that is BLAHBLAH/profile/USERID/certifications. Without understanding why it was bad, I wrote a whole python based monitor to send weekly updates on who has passed and who remains on the needs to finish for training assignments. When this monitor was learned about it turned out that the team who wrote this didn’t intend to allow others to know their completion status.



One of the first ways that this specific bug can manifest is when we get back too much data and decide to only minimally process/show some of that data. In the example the JSON returned has additional values that weren’t reported on the page’s display.



These additional values are what’s used to pull up the profile’s data.



So with that in mind, we can see that the key is to iterate through the ID numbers until we find another user’s profile info. (Original idea seems to be much more thought out than my manual approach.) <https://docs.cycubix.com/web-application-security-essentials/solutions/a5-broken-access-control/insecure-direct-object-reference-5>

