Bussiness Logic Error Cheatsheet

1. change the price with other price :100->50
2. change the price with nagative price :100->-100
3. change the price with other price by add nagative value: 100 ->(+-120)
4. change the price with other price by mult by 0.5: 100->(0.5*100)
5. Retrieving a Profile

```
For example, Jack’s profile can be fetched with id=1001 and if this value
changed to 1089 we get another user’s information. A scanner may go on and
change the value from 1001 to **‘1001** to find SQL injection, but not to 1089 and
would miss deducing that the application is vulnerable to authorization bypass. By changing the “
```

6. Shopping Cart

```
Let us consider an online store application where customers add items to their shopping cart. The
may make it possible for attackers to bypass the authentication processes to
directly log into the shopping cart application and avoid paying for “purchased” items.
```

7. **Review Functionality**

   - Some applications have an option where verified reviews are marked with some tick or it's mentioned. Try to see if you can post a review as a Verified Reviewer without purchasing that product.
   - Some app provides you with an option to provide a rating on a scale of 1 to 5, try to go beyond/below the scale-like provide 0 or 6 or -ve.
   - Try to see if the same user can post multiple ratings for a product. This is an interesting endpoint to check for Race Conditions.
   - Try to see if the file upload field is allowing any exts, it's often observed that the devs miss out on implementing protections on such endpoints.
   - Try to post reviews like some other users.
   - Try performing CSRF on this functionality, often is not protected by tokens

8. **Coupon Code Functionality**

   - Apply the same code more than once to see if the coupon code is reusable.
   - If the coupon code is uniquely usable, try testing for Race Condition on this function by using the same code for two accounts at a parallel time.
   - Try Mass Assignment or HTTP Parameter Pollution to see if you can add multiple coupon codes while the application only accepts one code from the Client Side.
   - Try performing attacks that are caused by missing input sanitization such as XSS, SQLi, etc. on this field
   - Try adding discount codes on the products which are not covered under discounted items by tampering with the request on the server-side.

9. **Delivery Charges Abuse **

   - Try tampering with the delivery charge rates to -ve values to see if the final amount can be reduced.

    - Try checking for the free delivery by tampering with the params.

10. **Currency Arbitrage **

- Pay in 1 currency say USD and try to get a refund in EUR. Due to the diff in conversion rates, it might be possible to gain more amount.

11. **Premium Feature Abuse **

- Try forcefully browsing the areas or some particular endpoints which come under premium accounts.
- Pay for a premium feature and cancel your subscription. If you get a refund but the feature is still usable, it's a monetary impact issue.
- Some applications use true-false request/response values to validate if a user is having access to premium features or not.
- Try using Burp's Match & Replace to see if you can replace these values whenever you browse the app & access the premium features.
- Always check cookies or local storage to see if any variable is checking if the user should have access to premium features or not.

12. **Refund Feature Abuse**

- Purchase a product (usually some subscription) and ask for a refund to see if the feature is still accessible.
- Try for currency arbitrage explained yesterday.
- Try making multiple requests for subscription cancellation (race conditions) to see if you can get multiple refunds.

13. **Cart/Wishlist Abuse **

- Add a product in negative quantity with other products in positive quantity to balance the amount.
- Add a product in more than the available quantity.
- Try to see when you add a product to your wishlist and move it to a cart if it is possible to move it to some other user's cart or delete it from there.

14. **Thread Comment Functionality**

- Unlimited Comments on a thread
- Suppose a user can comment only once, try race conditions here to see if multiple comments are possible.
- Suppose there is an option: comment by the verified user (or some privileged user) try to tamper with various parameters in order to see if you can do this activity.
- Try posting comments impersonating some other users.

15. **Parameter Tampering **

- Tamper Payment or Critical Fields to manipulate their values
- Add multiple fields or unexpected fields by abusing HTTP Parameter Pollution & Mass Assignment
- Response Manipulation to bypass certain restrictions such as 2FA Bypass

16. **Parameter tampering can result in product price manipulation**

- https://www.youtube.com/watch?v=3VMlV7j_yzg

17. **Manipulation of exam results at Semrush.Academy**

- In this situation, it was possible to bypass the exam process. That is to replace the results of the exam with the correct ones and send a request to get the certificate right away. And to replace the results with the correct ones turned out, because the body of the request was json, where 1 = true, and empty = false.

Steps To Reproduce:

1. Finished exams with any answers
2. Retake exam
3. Send the last request of our answer

Example body: `{"answers":
{"503":"","504":"","505":"1","506":"","507":"","591":"1","592":"","593":"","594":"","595":"","596":"","1340":"1","1341":"","1342":"","1343":"","1344":"","1345":"","1351":"1","1352":"","1353":"","1354":"","1355":"","1356":"","1357":"","1358":"1","1359":"","1360":"","1361":"","1362":"","1363":"","1365":"1"}}

18. **Authentication flags and privilege escalations at application layer.** Applications have their own access control lists (ACLs) and privileges. The most critical aspect of the application related to security is authentication. An authenticated user has access to
the internal pages and structures that reside behind the login section. These privileges can be maintained by the database, LDAP, file etc. If the implementation of authorization is weak, it opens up possible vulnerabilities. If these vulnerabilities are identified
during a test, then there is the potential for exploitation. This exploitation would likely include accessing another user's content or becoming a higher-level user with greater permissions to do greater damage ***How to test for this business logic flaw:***

```
• During the profiling phase or through a proxy observe the HTTP traffic, both request and respon
• POST/GET requests would have typical parameters either in name-value pair, JSON, XML or Cookies
the parameter and the value need to be analyzed.
• If the parameter name is suspicions and suggests that it has something to do with ACL/Permissio
target.
• Once the target is identified, the next step is evaluating the value, it can be encoded in hex,
should do some tampering and try to define its behavior with bit of fuzzing.
• In this case, fuzzing may need a logical approach, changing bit patterns or permission flags li
on. Some combination of bruteforcing, logical deduction and artistic tampering will help to decip
successful then we get a point for exploitation and end up escalating privileges or bypassing aut
```

19. **Critical Parameter Manipulation and Access to Unauthorized Information/Content.** HTTP GET and POST requests are typically accompanied with several parameters when submitted to the application. These parameters can be in the form of name/value pairs, JSON, XML etc. Interestingly, these parameters can be tampered with and guessed (predicted) as well. If the business logic of the application is processing these parameters before validating them, it can lead to information/content disclosure. This is another common business logic flaw that is easy to exploit
How to test for this business logic flaw:

```
• During the profiling phase or through a proxy, observe HTTP traffic, both request and response
• POST/GET requests would have typical parameters either in name-value pair, JSON, XML or Cookies
the parameter and the value need to be analyzed.
• Observe the values in the traffic and look for incrementing numbers and easily guessable values
• This parameter's value can be changed and one may gain unauthorized access.
In the above case we were able to access other users profiles
```

20. **Developer's cookie tampering and business process/logic bypass.** Cookies are an essential component to maintain state over HTTP. In many cases, developers are not using session cookies only, but instead are building data internally using session only variables. Application developers set new cookies on the browser at important junctures which exposes logical holes. After authentication logic sets several parameters based
    on credentials, developers have two options to maintain these credentials across applications. The developer can set the parameters in session variables or set cookies in the browser
    with appropriate values. If application developers are passing cookies, then
    they might be reverse engineered or have values that can be guessed/ deciphered. It can create a possible logical hole or bypass. If an attacker can identify this hole then they can exploit it with ease
    How to test for this business logic flaw:

```
• During the profiling phase or through a proxy observe the HTTP traffic, both request and respon
• Analyze all cookies delivered during the profiling, some of these cookies will be defined by de
session cookies defined by the web application server.
• Observe cookie values in specific, look for incrementing easily guessable values across all coo
• Cookie value can be changed and one may gain unauthorized access or logical escalation
```

21. ** LDAP Parameter Identification and Critical Infrastructure Access** LDAP is becoming an important aspect for large applications and it may get integrated with "single sign on" as well. Many infrastructure layer tools like Site Minder or Load Balancer use LDAP for both authentication and authorization. LDAP parameters can carry business logic decision flags and those can be abused and leveraged. LDAP filtering being done at the business application layer enable logical injections to be possible on those parameters. If the application is
    not doing enough validation then LDAP injection and business layer bypasses are possible.

How to test for this business logic flaw:

```
• During the profiling phase or through a proxy observe the HTTP traffic, both request and respon
• POST/GET requests would have typical parameters either in name-value pair, JSON, XML or Cookies
the parameter and the value need to be analyzed.
• Analyze parameters and their values, look for ON,CN,DN etc. Usually these parameters are linked
for the parameter taking email or usernames, these parameters can be prospective targets.
• These target parameters can be manipulated and injected with "*" or any other LDAP specific fil
can lead to logical bypass over LDAP and end up escalating access rights.
```

22. **Business Constraint Exploitation** The application's business logic should have defined rules and constraints that are very critical for an application. If these constraints are bypassed by an attacker, then it can be exploited. User
    fields that have poor design or implementation are often controlled by these business constraints. If business logic is processing variables controlled as hidden values then it leads to easy discovery and exploitation. While crawling and profiling the application, one can list all these possible different values and their injection places. It is easy to browse through these hidden fields and understand their context; if context is leveraged to control the business rules then manipulation of this information can lead to critical business logic vulnerabilities. ***How to test for this business logic flaw:***

```
• During the profiling phase or through a proxy observe the HTTP traffic, both the request and re
• POST/GET requests would have typical parameters either in name-value pair, JSON, XML or Cookies
the parameter and the value need to be analyzed.
• Analyze hidden parameters and their values, look for business specific calls like transfer mone
• These target parameters can be manipulated and values can be changed. It is possible to avoid t
```

23. **Business Flow Bypass** Applications include flows that are controlled by redirects and page transfers. After a successful login, for example, the application will transfer the user to the money transfer page.During these transfers, the user's session is maintained by a session cookie or other mechanism. In many cases,
    this flow can be bypassed which can lead to an error condition or information leakage. This leakage can help an attacker identify critical back-end information. If this flow is controlling
    and giving critical information out then it can be exploited in various use cases and scenarios ***How to test for this business logic flaw:***

- During the profiling phase or through a proxy observe the HTTP traffic, both request and respon
- POST/GET requests would have typical parameters either in name-value pair, JSON, XML or Cookies
- Identify business functionalities which are in specific steps (e.g. a shopping cart or wire tra
- Analyze all steps carefully and look for possible parameters which are added by the application
- These parameters can be tampered through a proxy while making the transaction. This disrupts th

24. ** Identity or Profile Extraction** A user's identity is one of the most critical parameters in authenticated applications. The identities of users are maintained using session or other forms of tokens. Poorly designed and
    developed applications allow an attacker to identify these token parameters from the client side and in some cases they are not closely maintained on the server side of the session as well. This scenario opens up a potential opportunity for abuse and system wide exploitation. The
    token is either using only a sequential number or a guessable username ***How to test for this business logic flaw:***

- During the profiling phase or through particular proxy observe HTTP traffic, both request and r
- POST/GET requests would have typical parameters either in name-value pair, JSON, XML or Cookies
- Look for parameters which are controlling profiles.
- Once these target parameters are identified, one can decipher, guess or reverse engineer tokens

25. **File or Unauthorized URL Access and Business Information Extraction Identity** Business applications contain critical information in their features, in the files that are exported and in the export functionality itself. A user can export their data in a selected file format (PDF, XLS or CSV) and download it. If this functionality is not carefully implemented, it can enable asset leakage. An attacker can extract this information from the application layer. This is one of the most common mistakes and easy to exploit as well. These files can be fetched directly
    from URLs or by using some internal parameters. ***How to test for this business logic flaw:***

- During the profiling phase or through a particular proxy, observe the HTTP traffic, both reques
- POST/GET requests would have typical parameters either in a name-value pair, JSON, XML or Cooki
- Identify file call functionalities based on parameter names like file, doc, dir etc. These para
- Once a target parameter has been identified start doing basic brute force or guess work to fetc

26. null pyloads
27. in change password try to delete current password