Logic Vulnerabilities

Riccardo BONAFEDE
Università di Padova

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Goal

- Learn how to find and address high level logic errors
- Learn how to analyse permissions in web applications
- Learn how to analyse functionalities and logic flaws in web applications

Prerequisites

- Lecture:
  - *WS_1.1 - HTTP Protocol and Web-Security Overview*

Outline

- Introduction
- Access-Control logic vulnerabilities
  - IDOR
- Incorrect input

Outline

- Introduction
- Access-Control logic vulnerabilities
  - IDOR
- Incorrect input

Introduction

- Business logic vulnerabilities are flaws in the design of an application that let users cause unintended behaviors
- These flaws often based on wrong assumptions about the user's behaviors

Types of Logic Vulnerabilities

- Logic Vulnerabilities are specific to the applications in which they occur
- The best way to group them is by the root cause:
  - Access Control Issues
  - Wrong assumption of the user behavior
  - Flawed handling of an Incorrect input
  - …

Outline

- Introduction
- Access-Control logic vulnerabilities
  - IDOR
- Incorrect input

Unprotected Functionality

- The most common type of access control issues are the one caused by a **wrong implementation** of the authentication system
- We call these type of issues «**Unprotected Functionality**»

Unprotected Functionality

- Unprotected Functionality flaws arise when the web application does not properly check user's privileges
- A trivial example is an administration *api* that does not enforce the privilege required by the user

Unprotected Functionality

- The best way to find these vulnerabilities is to try to use them without the required privilege
- For more complex web applications, with different privilege levels, is it possible to use an authentication matrix:
  - For every privilege level, create two or more users
  - For every user, test every functionality and save the result
  - Now, you can see the matrix created with the result, and it's easy to spot problems
- AuthMatrix[1] is a burp plugin to automate this type of testing

1: https://github.com/PortSwigger/auth-matrix

Unprotected Functionality

- It is also possible that the authentication is present, but is not enforced correctly
- This happens when the access control is not enforced by the functionality itself, but by some other system, for example by the web server itself

Unprotected Functionality

- For example, the functionality can be accessed in different ways, in which the authentication is not enforced

```
<Limit GET POST>
  Order Allow,Deny
  Deny from all
</LimitExcept>
```

Limit every request that is using a GET or POST method

Unprotected Functionality

- Route-based limiting can also be bypassed
  - Exploit path normalization
    - /api/v1/user_info == /api/v1/foo/**..;**/user_info
  - Framework-dependent behaviors
    - For example, in applications written using the *ruby on rails* web-application framework it is possible to append ".json" to a route. Often, developers forget to check the authentication on these paths

Outline

- Introduction
- Access-Control logic vulnerabilities
  - IDOR
- Incorrect input

Unprotected Functionality - IDOR

- IDOR – Insecure Direct Object Reference
  - One of the most common flaws in web applications
  - Arises when user input is used to access data directly and no authorization is enforced
  - Permits a horizontal privilege escalation

Unprotected Functionality - IDOR

- An example from “Vimeo.com Insecure Direct Object References Reset Password”[1]
- When resetting a password, Vimeo sends a reset link to the user email
- The link is in the form:
  - https://vimeo.com/forgot_password/[user_id]/[token]

1: https://hackerone.com/reports/42587

Unprotected Functionality - IDOR

- **`user_id`** is the id of the user that requests the password change
- **`token`** is a one time secret that Vimeo uses to verify the identity of the user that requests the password change
- What if the token was not linked to the user_id ?

Unprotected Functionality - IDOR

- Changing the user-id permits to change other users password, because the token is still valid

https://vimeo.com/forgot_password/1234567/7aaor252m19kjtkvtr

https://vimeo.com/forgot_password/9876543/7aaor252m19kjtkvtr

Unprotected Functionality - IDOR

- To find IDORs:
  1. Look at the request's parameters
  2. If you find something that can be a reference, try to change it

Outline

- Introduction
- Access-Control logic vulnerabilities
  - IDOR
- Incorrect input

Incorrect Input Handling

- Other types of logic flaws arise when a malicious user can provide a wrong input type to a web app
- A lot of developers assume that the input type can be enforced client-side

Incorrect Input Handling

For example, if a textbox is disabled doesn't mean that an attacker can send that parameter to the server

```
<form action="/buy">
  <input type="text" id="qnt" name="quantity">
  <input type="text" id="price" name="price" disabled>
  <input type="submit" value="Submit">
</form>
```

Incorrect Input Handling

- But, as you may know at this point, it is trivial to provide arbitrary values to the server by making a raw request
- If the web application does not handle this input correctly, then logic flaws may arise

Incorrect Input Handling

- A real-world example is provided in the report https://hackerone.com/reports/364843
- Using this kind of flaw, a researcher found a vulnerability in the site **upserve.com**, a *"Restaurant Management Software"*

Incorrect Input

- The vulnerable function works as follows:
  - A user add some items to its personal cart
  - Completed the order, the user can finalize the payment
  - The final price was calculated by summing the total cost of each different items multiplied the desired quantity

Incorrect Input

- The app was flawed; in fact, it didn’t expect a negative value in the quantity of an item
- Because of this, it was possible to get a “discount” on the final price
- To do so, the attacker needs to buy some products, then add a negative value to the quantity of one of the items

Incorrect Input

- If the quantity of a product is negative, then its cost is subtracted from the total
- This was possible because the web app didn't expect a negative value:
  - `"name": "BreadPudding", "price": 900, "quantity": -1`

Logic Vulnerabilities

Riccardo BONAFEDE
Università di Padova

https://cybersecnatlab.it