IDOR (Insecure Direct Object Ref)

- Relatively common vulnerability in web applications, easy to understand, detect, exploit and relatively easy to prevent
- CWE:
  - CWE-863: incorrect authorization → the product performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check. This allows attackers to bypass intended access restrictions
  - CWE-639: authorization bypass though user-controlled key → the system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data
- IDOR exploitation
  - Injection: remote with no user interaction
  - Impact: confidentiality → attacker can read data it should not be allowed to read (it slowly and systematically reads lots of data)
- IDOR-risky webapps
  - Many objects (database records, documents, files)
  - Objects identified by HTTP request parameters
  - Authentication → each user can access some objects but not all
- What happens correctly

```
1. Username := Extract from session
2. IF       Session is not authenticated
3. THEN     HTTP Response := "Error: please authenticate"

4. URL, params := Extract from HTTP Request
5. ObjectId := Determine based on URL, params

6. IF       Username can access ObjectId
7. THEN     HTTP Response := Read ObjectId from Database
8. ELSE     HTTP Response := "Unauthorized for this item"
```

-A relatively common weakness is l'assenza del punto 6, non viene controllato se lo user ha l'accesso a quell'oggetto, glielo si fa leggere e basta
-PHP in a nutshell: is a scripting language server side for dynamiting web document, the files on a web server are mixed HTML and PHP
-When receiving an HTTP request for URL-x, the PHP is executed (whose output is HTML), the PHP code is replaced with its output and then is retuned the HTML content
-A common mistake is the no authorization check (c'è l'immagine nelle slide con il codice, praticamente quando c'è il parametro invoice non c'è l'autenticazione sembrerebbe)

- Authenticated user (web attacker) interact with web app to analyze the structure of the web page and/or HTTP request, finds a promising parameter ?invoice=… → send many HTTP requests with different values of invoice and check the HTTP responses with error code
- At this point the web attacker can read all the objects for which it can generate the correct value for invoice, even those it is not allows to read
- Examples…
- Saltzer and Schroeder: complete mediation → every access to every object must be checked for authority
  - IDOR is ultimately a failure to apply this principle
- Esempio 18app: si può modificare il campo IDbeneficiario senza alcun tipo di controllo, quindi è possibile creare buoni utilizzando un account altrui…vedendo tutti i dati dell'account utilizzato (di fatto si usa il codice fiscale dell'altra persona per accedere alla sua pagina 18app)
- IDOR clues worth investigating
  - Request parameters name resembles the name of a database table/column
  - Parameter values easily guessed: usually incremental integer, the identifier can be obtained from other pages and can be constructed/ enumerated
- Italy 2023 esempio: IDOR dati sanitari: chiunque dopo aver superato le procedure di autenticazione informatica presenti nell'ambito del portale MyCivis, poteva visualizzare, selezionare e aprire uno o più documenti presenti nel FSE di un altro assistito, anche in assenza di delega, semplicemente inserendo nel predetto parametro patientID il codice fiscale di tale assistito. Quali fossero i test di sicurezza incombenti sul fornitore e per quali ragioni tali test pur regolarmente svolti non abbiano permesso di intercettare il bug, lo chiarisce la società (puntini puntini), questa sostiene di aver realizzato i vulnerability assessment, ma che ciò non sia bastato a evitare la presenza di questo bug
- Remarks del prof: the very same mistake as in 18app, the vulnerability assessment has not detected this mistake
  - The citizen who detected this vuln and notified the company has ben denunciato male and is currently being investigated

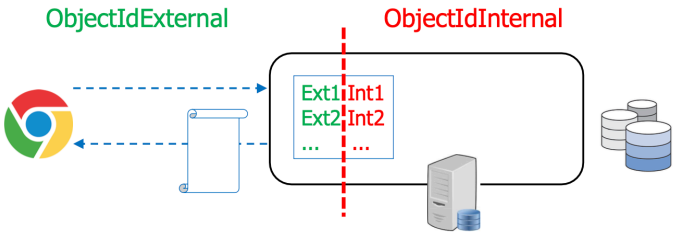
Webapp defense in general

- Prevention: attempt to not write vulnerable code
  - Programming methodology: it requires competent and disciplined developers, must be applied correctly and systematically. Can hardly be applied on the entire code base (legacy, framework, plugins)
  - NO SYSTEMATIC APPROACH: JUST BROAD RULES
- Mitigation: attempt to make exploitation very difficult/impossible
  - HTTP options: can be activated on the web server, supported by all modern browsers
  - Code independent or minimal code changes
  - NO DEDICATED HTTP SUPPORT
- Defense in depth: always use both strategies

IDOR prevention

- Numero 6 dello pseudocodice sopra: if Username can access ObjectID is a fundamental check → authorization with design principle of the complete mediation
- Predefined good links: the web app W generates pages where links to W are only to objects that the user can access
  - Generates pages where links to W are constructed dynamically by javascript code in the browser
  - Javascript code constructs only URLs corresponding to the current user
  - **NO: CLIENT CODE OR BEHAVIOR CANNOT BE TRUSTED**, the client can construct every name-value pair it wishes
- Obfuscated links: the web app W generates pages where links to W that identify objects are very complex (many parameters with strange names) and parameters values that are very long
  - **NO: LONG AND STRANGE IS NOT EQUAL TO NOT ENUMERABLE OR NOT PREDICTABLE**
- Implicit (wrong) assumptions
  - Browser will only follow the links that we prepare for it
  - Users will only make a few attempts to manipulate links, if at all
  - *NB CLIENT CODE OR BEHAVIOR CANNOT BE TRUSTED*
- IDOR prevention in a nutshell:

1. Implement authorization check systematically + construct URLs with parameter values (not predicable and not enumerable)



- In pratica l'ObjectID nel URL (che quindi lo user vede) idealmente dovrebbe essere l'hash dell'objectID salvato nel database interno al server, in modo da non essere predictable or enumerable
- In realtà basterebbe non saltare il passaggio dell'autorizzazione: se viene fatto il controllo sull'accesso alle risorse, allora non è fondamentale che gli URL siano con parametri non predictable e non enumerable (ma meglio più sicuri che meno sicuri)
- Quindi implementare solo la seconda fase, senza implementare una forma ti authorization check prima, non è una buona idea, ma nel caso in cui sia davvero difficult e costly implementare il check, mentre gli URL sono davvero non predictable e non enumerable, allora può essere un trade-off accettabile (meglio di niente)
- Implementation outline

❑ Table ObjectIdExternal - ObjectIdInternal

❑ Request handling
  ❑ Internal := TableLookup(External)

❑ Response handling
  ❑ External := Hash(Internal)
  ❑ TableInsert(External, Internal)

-HTTP request: objectsIDs can be placed in requested URL, query string in requested URL, query string in transmitted resource (POST), transmitted resource (POST, json resource)
-HTTP response: objectsIDs (to be used in future requests) can be placed in transmitted resource (HREF attributed of HTLM documents), transmitted resource (javascript variables, json resource)…
-Implementation detail: table objectIDexternal - objectIDinternal is better one per use than one per web app, URL history may contain objectIDs, and the attacker might be able to access URL history of some users