

Broken Access Control

OWASP Top 10 - Series

Presenter – Jalaj
@ null-meet, Airtel Centre, Gurgaon
27 August 2022

>_ whoami

- Tech Enthusiast
- Cyber security geek
- Ex-Suzuki, Ex-Dcm Hyundai (Mech.)
- Codes in C++ and Python. Exploring Go
- CTF Player
- CRAC research group : CVE Analysis and Cloud Security
- Learning about Cloud security and Malware analysis
- Bikes
- MMA



01

Introduction
to
OWASP Top 10

03

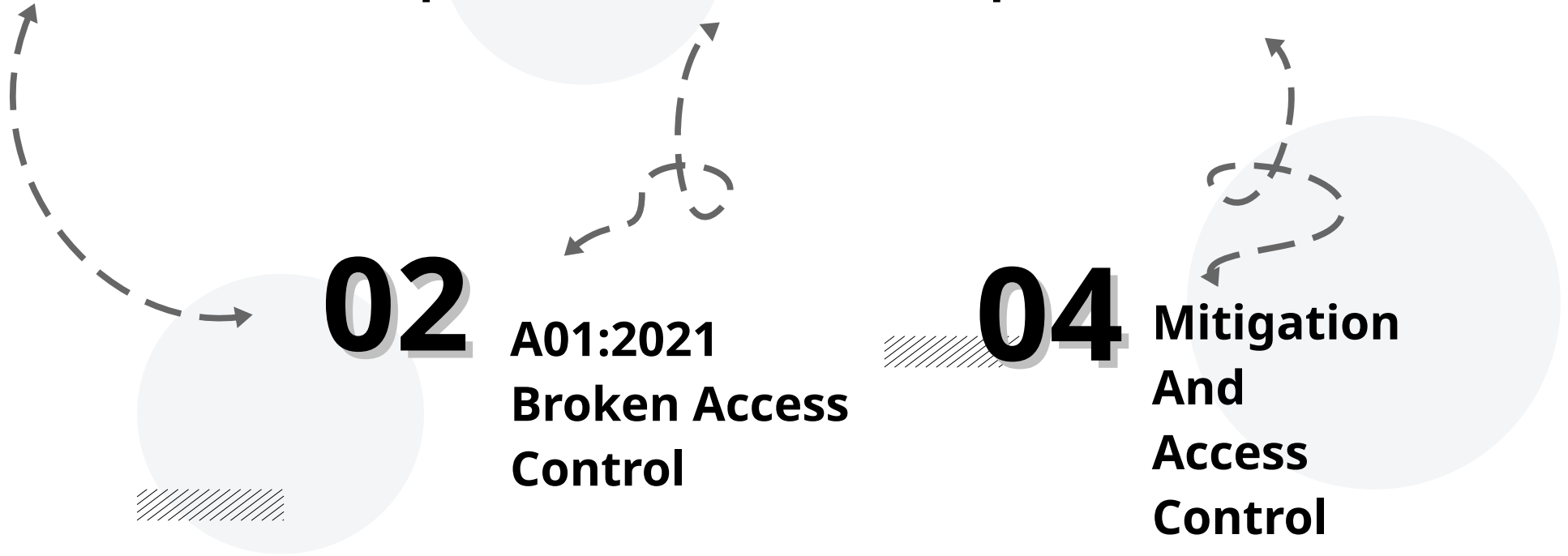
Vulnerabilities
With
Examples

02

A01:2021
Broken Access
Control

04

Mitigation
And
Access
Control
Mechanisms



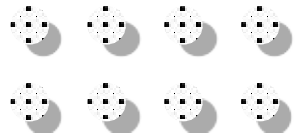
OWASP Top 10

The OWASP Top 10 is a de facto industry standard that provides a list of the 10 Most Critical Web Application Security Risks

- Broken Access Control
- Cryptographic Failure
- Injection
- Insecure Design
- Security Misconfiguration
- Vulnerable and Outdated Components
- Identification and Authentication Failures
- Software and Data Integrity Failures
- Security Logging and Monitoring Failures
- Server-Side Request Forgery (SSRF)




Illustrations by Pixeltrue on
[icons8](#)






Broken Access Control

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.



Broken Access Control moves up from the fifth position; 94% of applications were tested for some form of broken access control. The *34 Common Weakness Enumerations (CWEs)* mapped to Broken Access Control had more occurrences in applications than any other category.



List of Mapped CWEs

CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

CWE-23 Relative Path Traversal

CWE-35 Path Traversal: '.../...//'

CWE-59 Improper Link Resolution Before File Access ('Link Following')

CWE-200 Exposure of Sensitive Information to an Unauthorized Actor

CWE-201 Exposure of Sensitive Information Through Sent Data

CWE-219 Storage of File with Sensitive Data Under Web Root

CWE-264 Permissions, Privileges, and Access Controls (should no longer be used)

CWE-275 Permission Issues

CWE-276 Incorrect Default Permissions

CWE-284 Improper Access Control

CWE-285 Improper Authorization

CWE-352 Cross-Site Request Forgery (CSRF)

CWE-359 Exposure of Private Personal Information to an Unauthorized Actor

CWE-377 Insecure Temporary File

CWE-402 Transmission of Private Resources into a New Sphere ('Resource Leak')

CWE-425 Direct Request ('Forced Browsing')

CWE-441 Unintended Proxy or Intermediary ('Confused Deputy')

CWE-497 Exposure of Sensitive System Information to an Unauthorized Control Sphere

CWE-538 Insertion of Sensitive Information into Externally-Accessible File or Directory

CWE-540 Inclusion of Sensitive Information in Source Code

CWE-548 Exposure of Information Through Directory Listing

CWE-552 Files or Directories Accessible to External Parties

CWE-566 Authorization Bypass Through User-Controlled SQL Primary Key

CWE-601 URL Redirection to Untrusted Site ('Open Redirect')

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-651 Exposure of WSDL File Containing Sensitive Information

CWE-668 Exposure of Resource to Wrong Sphere

CWE-706 Use of Incorrectly-Resolved Name or Reference

CWE-862 Missing Authorization

CWE-863 Incorrect Authorization

CWE-913 Improper Control of Dynamically-Managed Code Resources

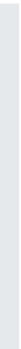
CWE-922 Insecure Storage of Sensitive Information

CWE-1275 Sensitive Cookie with Improper SameSite Attribute

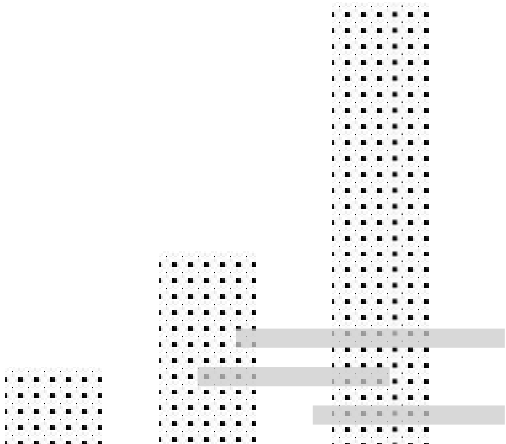
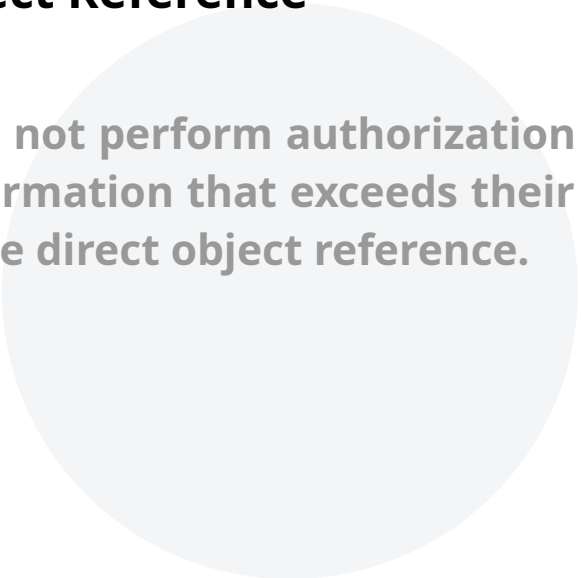


IDOR

Insecure Direct Object Reference



If the application does not perform authorization checks, the user may be permitted to view information that exceeds their authority. This situation is known as an insecure direct object reference.



IDOR can be found in 3 places

01 Query Component

Changing Query component: id=24

```
https://website.com/profile?id=23
```

02 Post Parameters

Form updates user password

```
<form method="POST" action="/update-password">  
  <input type="hidden" name="user_id" value="123">  
  <div>New Password:</div>  
  <div><input type="password" name="new_password"></div>  
  <div><input type="submit" value="Change Password">  
</form>
```

03 Cookies

Cookies should be random strings with high entropy

```
GET /user-information HTTP/1.1  
Host: insecure-website.com  
Cookie: user_id=9  
User-Agent: Mozilla/5.0 (Ubuntu;Linux) Firefox/101.0
```

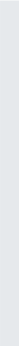

Preventing IDOR Vulnerability

- Enforce Strong Access Control over resources. Like resources mapped to the owner with some sort of random non-guessable number or non-sequence keys.
- Check the user authorization before issuing the resource when the user requests it. A strong User role function should be implemented in the application. Validate the user role and user hash with the resource before the resource is released.
- Don't map the object with a direct id instead use of hash is handy in this case. To make more difficult use of salt along with hash is good.

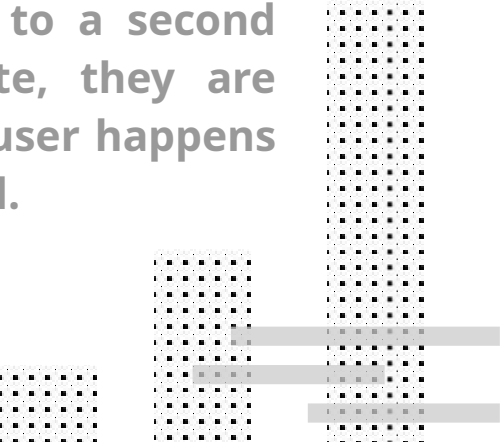


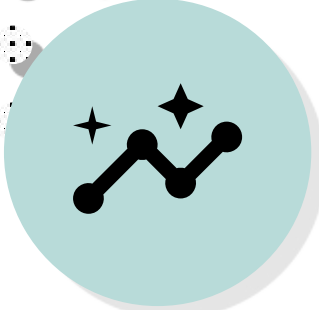
CSRF

Cross-Site Request Forgery



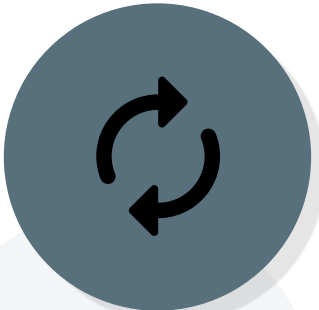
XSRF attacks work by making the reasonable assumption that users are often logged into many different websites at the same time. Attackers then embed code in one website that sends a command to a second website. When the user clicks the link on the first site, they are unknowingly sending a command to the second site. If the user happens to be logged into that second site, the command may succeed.



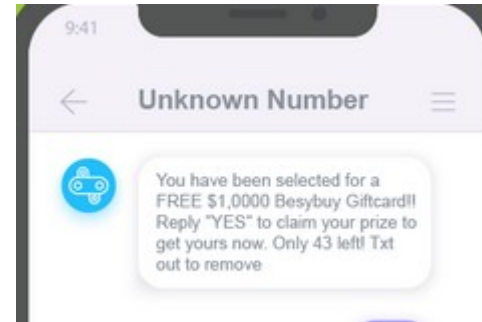


**Attacker hosts website that
calls original website with
HTML tag**

```
<img src = "https://samplebank.com/  
onlinebanking/transfer?  
amount=5000&accountNumber=425654"  
width="0" height= "0">
```



**Victim clicks on the
forged request (image)**



**Forged request is sent from
authorised session of victim**



Preventing CSRF Vulnerability

- Using a good anti CSRF token with high entropy
- Identifying where request is coming from
- Unique token for each session

Path Traversal

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The `'.../.../'` manipulation is useful for bypassing some path traversal protection schemes. If `"../"` is filtered in a sequential fashion, as done by some regular expression engines, then `".../.../'"` can collapse into the `"../"` unsafe value (CWE-182). Removing the first `"../"` yields `".../'"`; the second removal yields `"../"`. Depending on the algorithm, the software could be susceptible to CWE-34 but not CWE-35, or vice versa.

01 Imagine an online Image Portal saving files with names - 218.png

02 On server file is saved in a directory

```
/loadImage?filename=218.png  
  
/var/www/images/218.png
```

03 Directory structure in unix based systems

```
https://insecure-website.com/  
loadImage?filename=../../../../  
etc/passwd  
  
/var/www/images/../../../../  
etc/passwd  
  
/etc/passwd
```

04 If filters are in place then encode

```
filename=/etc/passwd  
  
....// or ....\  
  
..%c0%af or ..%252f
```

Preventing Path Traversal Vulnerability

1. Application should validate User Input

- Whitelist/Allow list of permitted values
- Permitted characters

2. Application should append the User Input to and use a platform dependent API

- Canonicalize the Path
- Verify the expected base directory

Access Control Concepts

- Enforce access control by an activity or feature, not the role
- Implement data-contextual access control to assign permissions to application users in the context of specific data items for horizontal access control requirements
- Build a centralized access control mechanism
- Design access control so all requests must be authorized
- Deny by default
- Server-side trusted data should drive access control policy decisions
- Build grouping capability for users and permissions
- Build admin screen first to manage access control policy data

Further reading resources & links



Scan :



Thanks!

Any Questions ?

