ASSIGNMENT COVERSHEET

UTS: ENGINEERING & INFORMATION TECHNOLOGY

SUBJECT NUMBER & NAME	NAME OF STUDENT(s) (PRINT CLEARLY)	STUDENT ID(s)
41182 System security	*Ryea D'souza* *Shayan Ali* *SURNAME* *FIRST NAME*	25473176 25555501

STUDENT EMAIL	STUDENT CONTACT NUMBER
Ryea.f.dsouza@student.uts.edu.au Shayan.ali@student.uts.edu.au	0480240944 0405124796

NAME OF TUTOR	TUTORIAL GROUP	DUE DATE
Calvin	3	19/05/25

ASSESSMENT ITEM NUMBER & TITLE
Assessment 2- project report

☐ I acknowledge that if AI or another nonrecoverable source was used to generate materials for background research and self-study in producing this assignment, I have checked and verified the accuracy and integrity of the information used.
☐ I confirm that I have read, understood and followed the guidelines for assignment submission and presentation on page 2 of this cover sheet.
☐ I confirm that I have read, understood and followed the advice in the Subject Outline about assessment requirements.
☐ I understand that if this assignment is submitted after the due date it may incur a penalty for lateness unless I have previously had an extension of time approved and have attached the written confirmation of this extension.

Declaration of originality: The work contained in this assignment, other than that specifically attributed to another source, is that of the author(s) and has not been previously submitted for assessment. I have rewritten any material provided by AI or other nonrecoverable sources and where appropriate acknowledged their contribution. I understand that, should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with University policy and rules. In the statement below, I have indicated the extent to which I have collaborated with others, whom I have named.

No content generated by AI technologies or other sources has been presented as my own work and I have rewritten any text provided by AI or other sources in my own words.

Statement of collaboration:
We collaborated and completed the assessment together

Signature of student(s) _____RD, SA_____ **Date**|
19/05/25_____

System Security- Broken Access Control

Assessment 2

By: Ryea & Shayan

Table of Contents

Abstract

Problem Statement

Broken Access Control (BAC) is one of the most critical security risks in modern web applications, ranked as A01 in the OWASP Top 10 (2021). It occurs when systems fail to properly restrict authenticated users, allowing them to access data, functions, or accounts they shouldn't. Attackers often exploit this by tampering with parameters or using forced browsing to bypass permission checks, which can lead to unauthorised access, data modification, or privilege escalation. These flaws usually stem from weak or missing access control measures caused by poor design, misconfiguration, or coding mistakes. Preventing BAC requires strong server-side checks and applying principles like Least Privilege to ensure users can only access what aligns with their role.

Main Findings

This report demonstrates the exploitation of a Broken Access Control (BAC) vulnerability, specifically an Insecure Direct Object Reference (IDOR), within the WebGoat web application using the Burp Suite web proxy tool. The attack showcased how an attacker can manipulate user-identifiable parameters in HTTP requests to gain unauthorised access to another user’s sensitive data. Key findings highlight that predictable resource identifiers and a lack of proper server-side authorisation checks are major contributors to such vulnerabilities.

As WebGoat is designed for educational purposes, it does not always reflect real-world environments or safeguards, so its results cannot be directly applied. Therefore, strong server-side validation, proper handling of IDOR, adherence to the Principle of Least Privilege (PoLP), and the use of Role-Based Access Control (RBAC) are recommended. Employing multiple security measures is essential to effectively mitigate BAC vulnerabilities and protect system data.

Introduction

Chosen Vulnerability: Broken Access Control

Access control is used to determine what a user can do after being authenticated in a system. If the rules for access control are not properly put in place, users may be able to access resources or data that they shouldn't be able to see. According to OWASP, BAC is the leading risk because it is so common and can seriously affect the security of web applications (OWASP, 2021). With modern applications becoming more complex and access rules often being inconsistent, these issues are widespread.

Attack Methods

Missing Authorisation Checks After Login

Broken Access Control usually happens when a system correctly identifies a user but does not verify their access rights. This occurs when users can access pages, functions or data without having their roles or permissions checked. An attacker may try to access restricted content by changing the values in URLs or forms. One of the results is that unapproved access to admin areas or private details could lead to changes in settings, data being leaked or losing control over important systems.

Insecure Direct Object References

IDOR occurs when users can directly manipulate object identifiers (like user IDs or invoice numbers) in the URL or request to access someone else's data. For example, changing ?invoice_id=101 to ?invoice_id=102 without access validation reveals another user's invoice. Attackers use URL tampering or API requests to probe such weaknesses (Portswigger, n.d.). This leads to data privacy breaches, especially when Personally Identifiable Information (PII) or financial data is exposed or modified without authorisation.

Privilege Escalation via Token or Cookie Manipulation

In certain systems, the roles or levels of access for users are held in cookies or tokens sent from the client. If the server does not properly validate these, someone could change role=user to role=admin. It is referred to as cookie manipulation or token tampering. The results can be significant because the attacker may control restricted areas or operations or manipulate important data (OWASP, 2013).

Unprotected Endpoints or APIs

APIs are used by many applications to access backend functions, but if these APIs are not checked for access, they can be attacked. Attackers could make unauthorised actions by forcing browsing or sending requests with PUT, DELETE, or POST methods. This happens when the back-end endpoints are not always secure, which enables attackers to perform actions like deleting data, updating permissions, or stealing a lot of sensitive information (Akamai, 2024).

Missing Function-Level Access Control

Some applications rely only on the user interface to hide admin features from regular users, but forget to enforce these restrictions on the server side. Attackers bypass the UI and manually send HTTP requests to access hidden functions. This is often done using tools that intercept and replay requests. If access checks are missing, attackers can run admin-level commands, modify settings, or perform operations they should not be authorised to carry out, which can affect data integrity and system security.

Defense Methods

Principle of Least Privilege

The Principle of Least Privilege makes sure that users and components do not have more permissions than necessary for their roles. As a result, users are restricted to using the permissions and data assigned to their role. Due to this principle, hackers are less likely to access the admin area or private information by using stolen accounts (What Is the Principle of Least Privilege?, n.d.).

Role-Based Access Control

RBAC makes it possible to control access by giving each user a specific role with clearly outlined permissions. By using this method, only users with the right role can access and use protected endpoints and functions. It is particularly important for avoiding risks such as function-level access control not being in place and direct access to admin features (Role-Based Access Control, 2022).

Server-Side Enforcement of Access Control

Server-side enforcement means that access checks are performed by the system, not only in the frontend. As a result, attackers cannot use direct requests or tools to circumvent security measures. It is necessary to protect against issues such as IDOR, forced browsing, and missing backend authorisation checks.

Indirect Object References

Using indirect object references makes it harder for attackers to guess or manipulate identifiers like user IDs or file paths. This directly mitigates IDOR vulnerabilities by hiding internal structure and preventing unauthorised access through parameter tampering or URL manipulation.

Logging, Monitoring, and Regular Testing

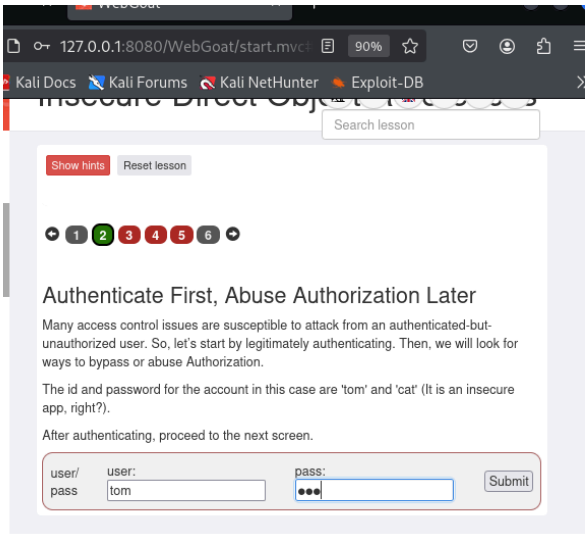
Logging and monitoring help detect abnormal behavior, such as repeated access attempts or unauthorised requests. Regular testing, including penetration testing and code reviews, helps identify weaknesses like exposed APIs or improper access control logic. These practices strengthen overall security and ensure known vulnerabilities like unprotected endpoints or privilege misuse are caught early.

Body

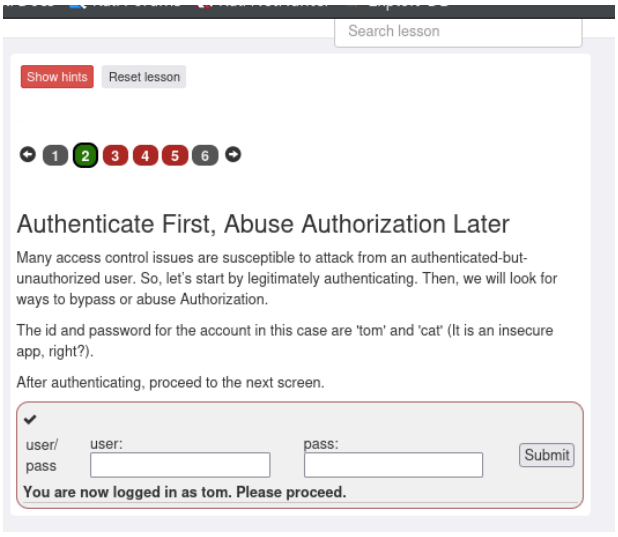
Attack Methods (Ryea)

Insecure Direct Object References: By altering unique identifiers (IDs) in URLs, attackers can get unauthorised access to resources that are blocked in several systems due to critical vulnerabilities known as IDORs. This vulnerability allows hackers to bypass access barriers and retrieve or change private data, which can have detrimental effects like unauthorised access to data, compromised sensitive data, and improper use of system functionalities. To protect user privacy, data integrity, and system security as a whole, these vulnerabilities must be fixed.

The following example shows how serious security breaches and privacy violations can result from the combination of Broken Access Control and IDOR.

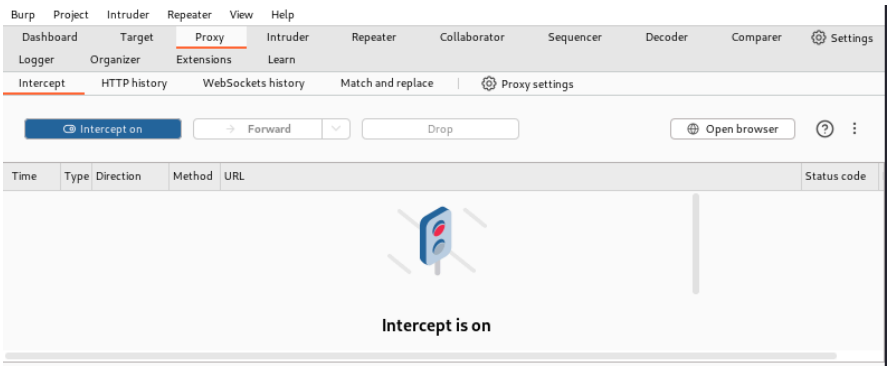


[Figure 1- WebGoat]



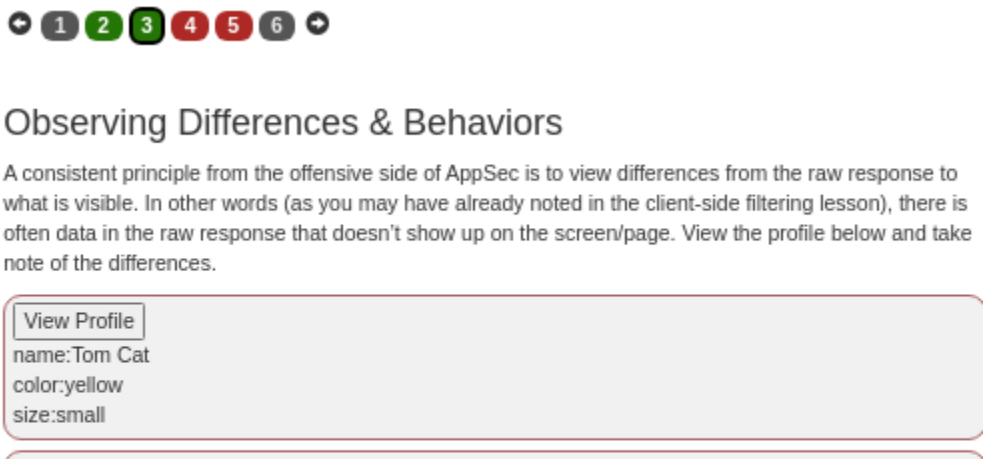
[Figure 2- WebGoat]

- Figure 1 shows a login screen where the user is instructed to log in with the username "tom" and password "cat". This demonstrates the authentication step.
- Figure 2 shows that the login and authentication were successful



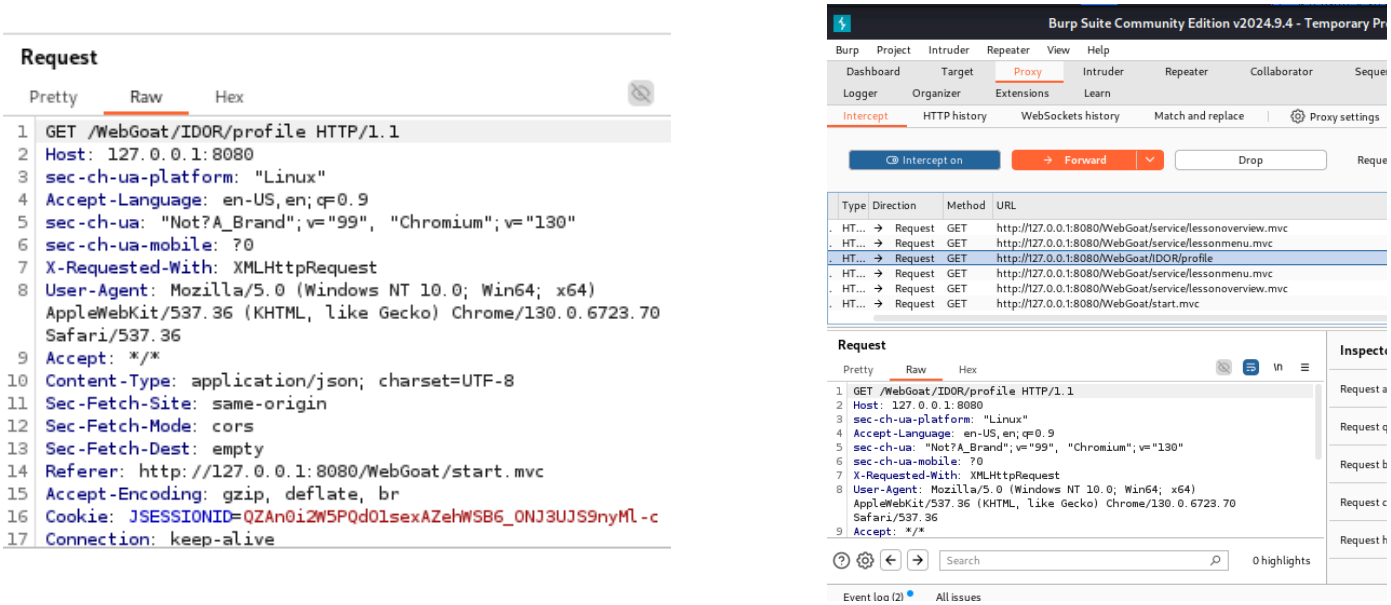
[Figure 3- WebGoat]

Figure 3 shows Burp Proxy with the intercept function turned on, ready to capture HTTP requests



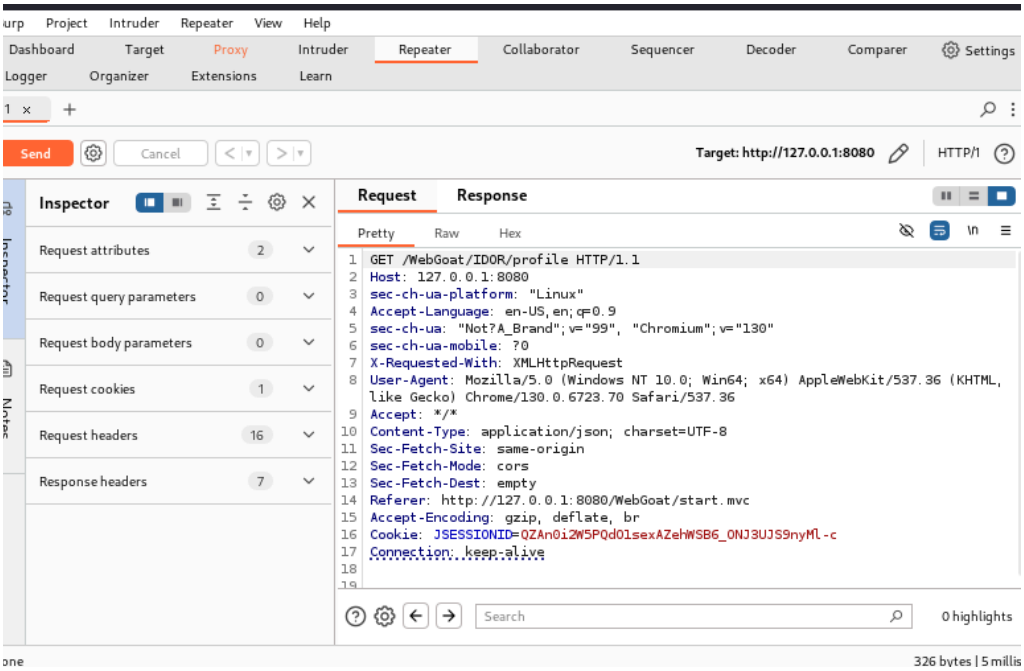
[Figure 4- WebGoat]

Once the intercept is on, go back to WebGoat and click View Profile. This ensures that the request for the profile data was captured in Burp Proxy.



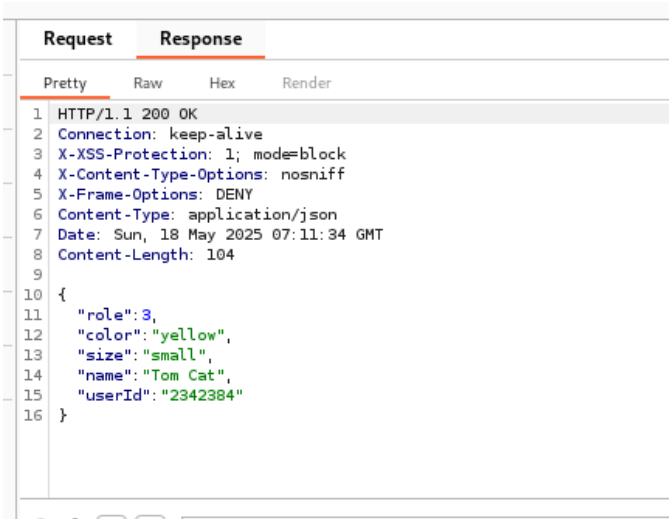
[Figure 5- WebGoat]

Click forward a few times until the request is found, in this case, it is the profile endpoint, and then forward the request to the repeater tool



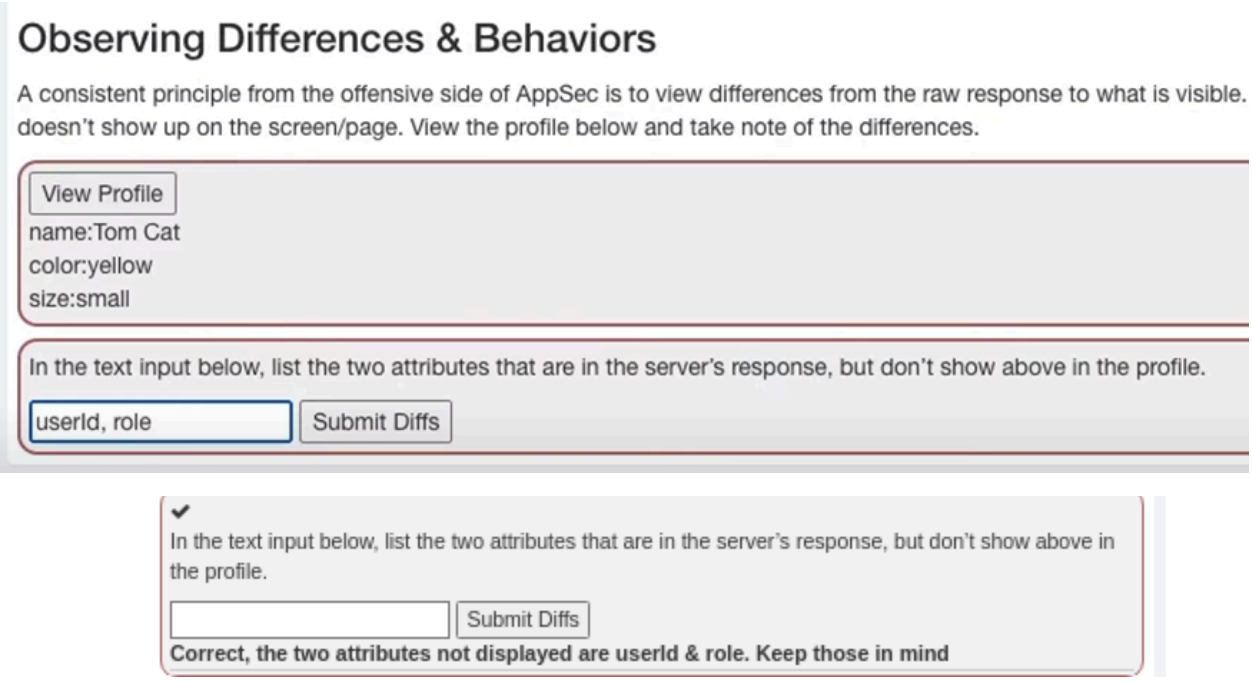
[Figure 6- WebGoat]

Navigate to the repeater tool and send the response again



[Figure 7- WebGoat]

In the response, it reveals the color, size, and name as well as two new fields, role and userid



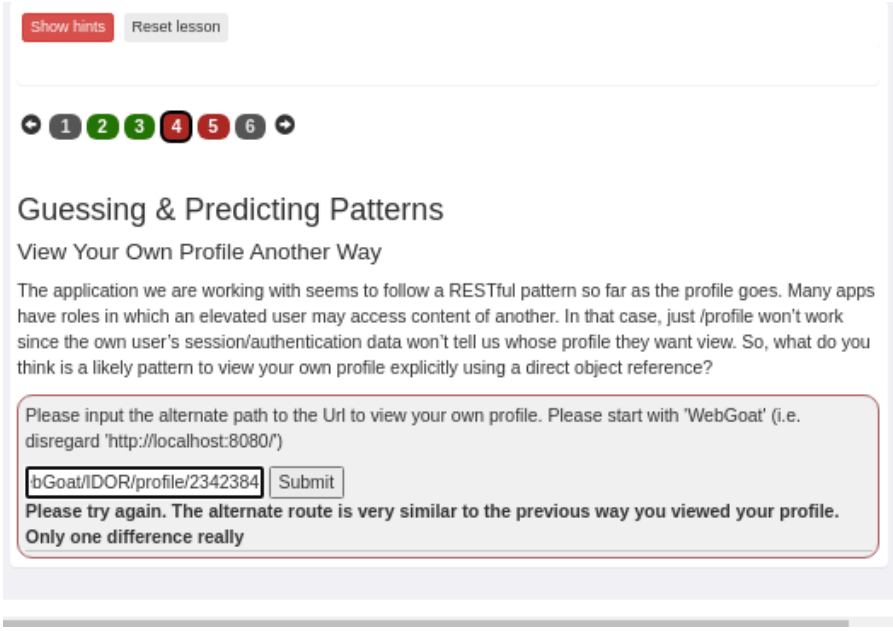
[Figure 8- WebGoat]

Figure 8 shows the initial analysis of a user profile response, where hidden attributes like userId and role are present in the server's response but not displayed on the user interface.

```
GET /WebGoat/IDOR/profile/2342384 HTTP/1.1
Host: 127.0.0.1:8080
sec-ch-ua-platform: "Linux"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
sec-ch-ua-mobile: ?0
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70
Safari/537.36
Accept: */*
Content-Type: application/json; charset=UTF-8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8080/WebGoat/start.mvc
```

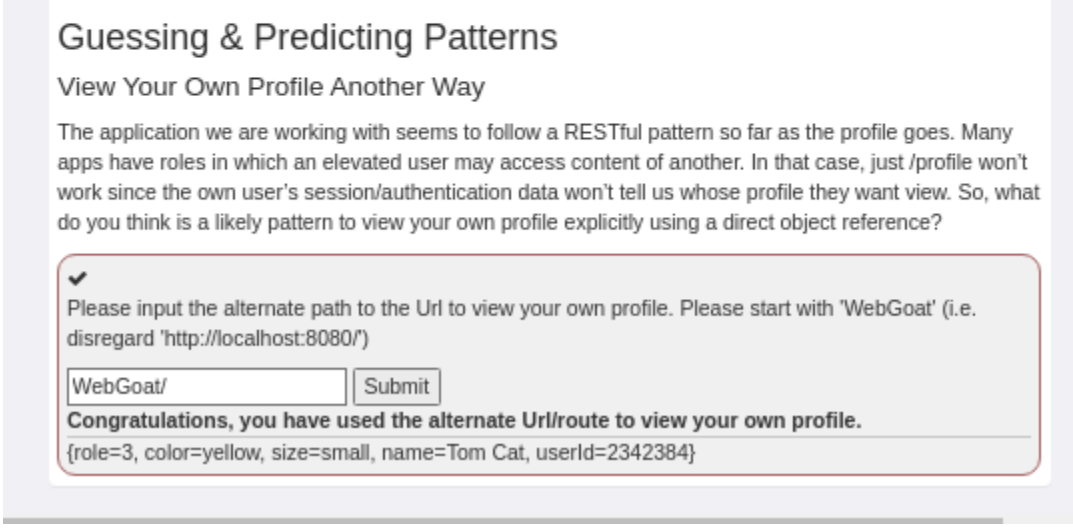
[Figure 9- WebGoat]

Copy the URL after the GET method and paste this alternate path into the search box after clearing the box



[Figure 10- WebGoat]

Place a slash after the word profile and then paste the userId (e.g /WebGoat/IDOR/profile/2342384)

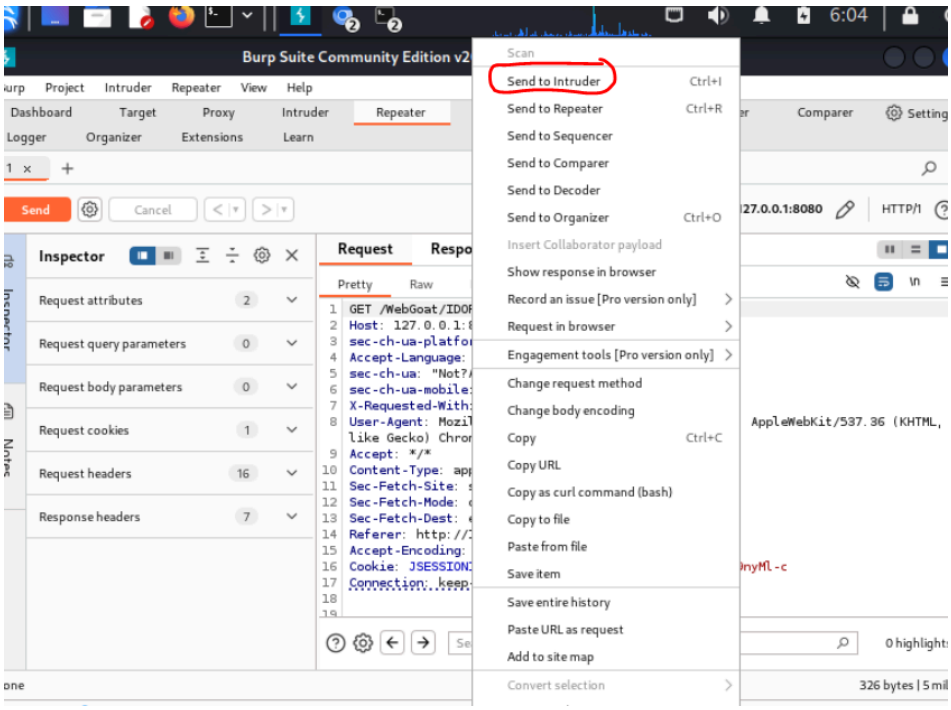


[Figure 11- WebGoat]

By clicking "Submit," it should enable the user to view the profile using a different URL or route.

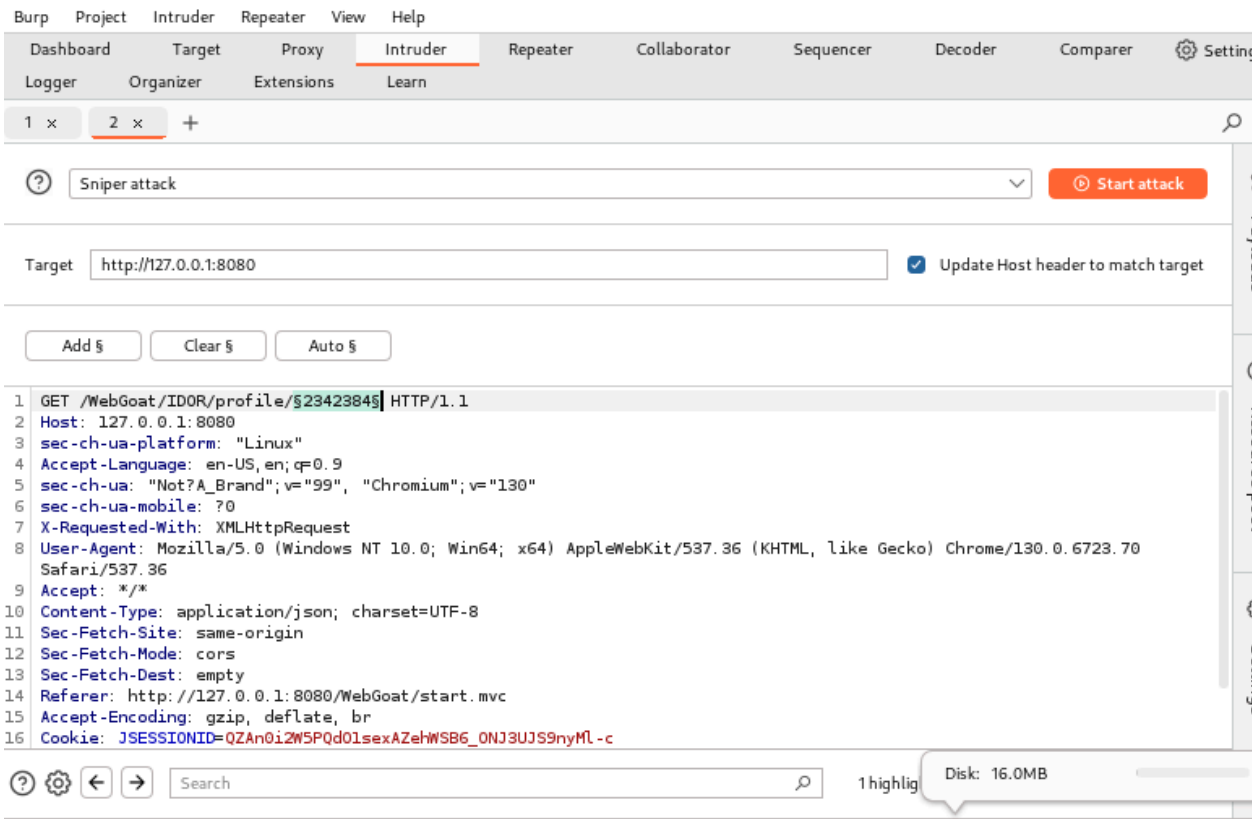
This demonstrates a BAC vulnerability, specifically an IDOR, by showing how a user can manually construct a URL containing a user-specific identifier to access profile information. It highlights how an attacker, after authenticating as a normal user, can observe the structure of a legitimate request, extract their userId, and then predict the format of the URL used to fetch user profiles. By directly entering a path like /WebGoat/IDOR/profile/2342384, the system returns user-specific data without properly verifying authorisation, proving that server-side access checks are either missing or insufficient.

To identify and access another user's profile, the attack involved testing a range of values to discover valid identifiers linked to other user accounts.



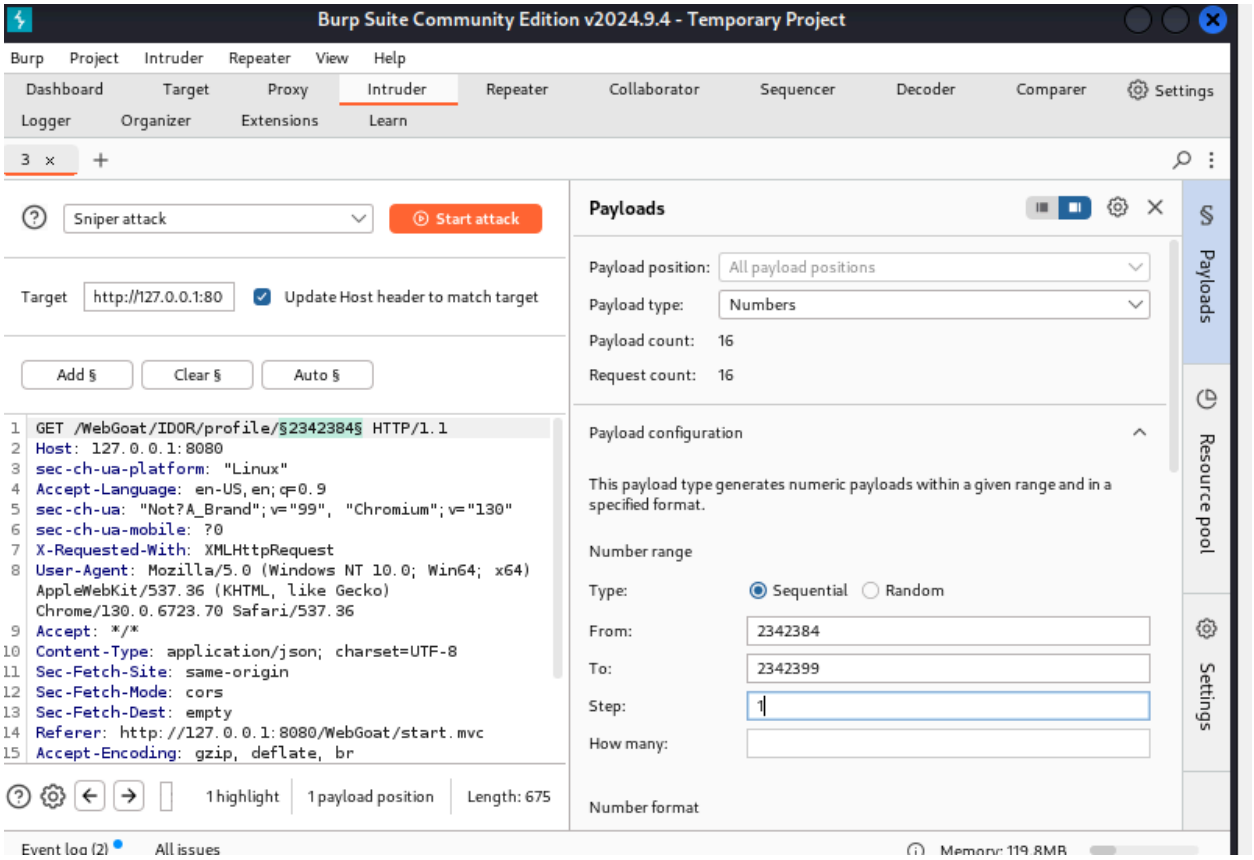
[Figure 12- WebGoat]

Take the request from the repeater for Tom's profile and forward it to the intruder



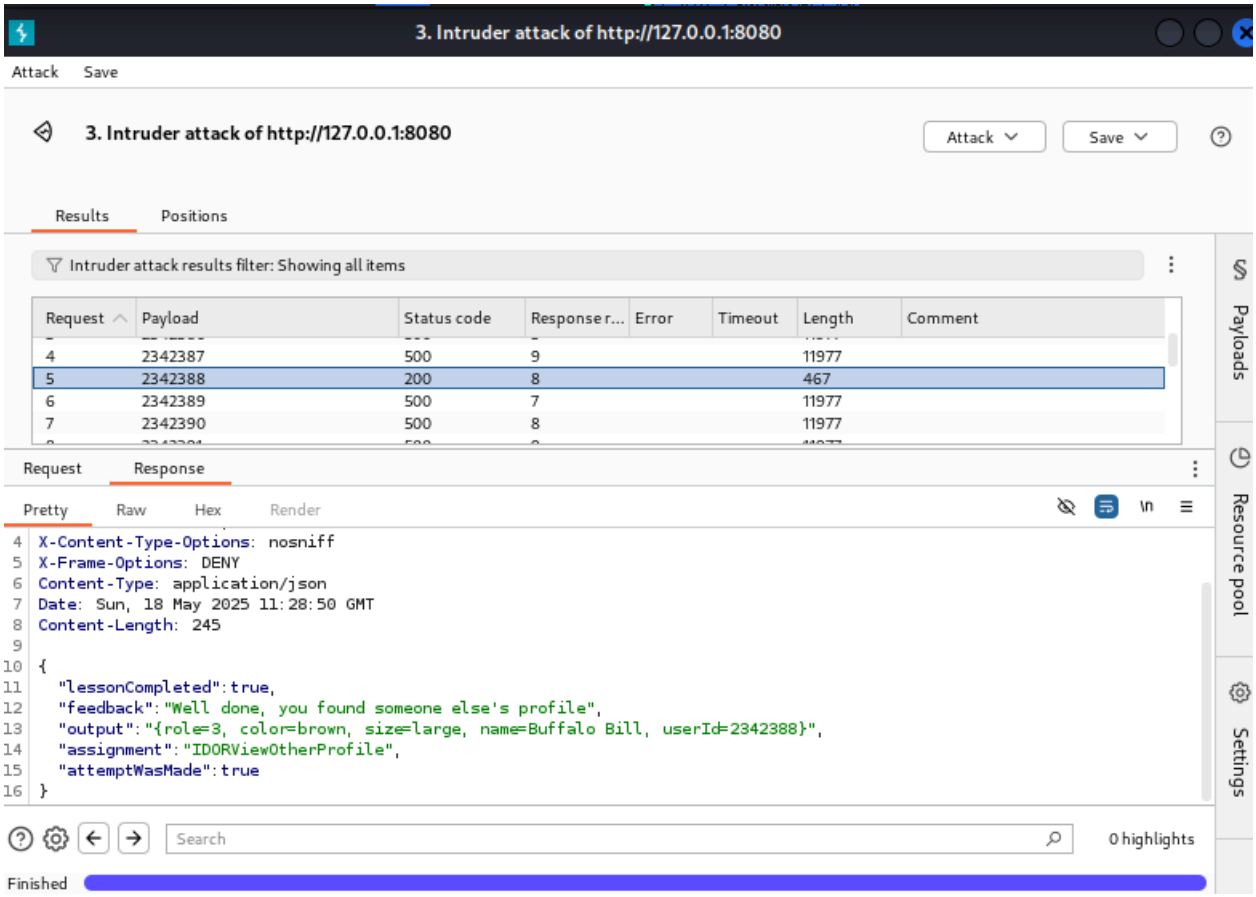
[Figure 13- WebGoat]

Navigate to the Intruder tool and clear all payload markers ($), and then highlight the userId and click on add payload markers



[Figure 14- WebGoat]

- Configure the payload settings by clicking on the "Payloads" tab and then selecting "Numbers" as the payload type.
- Configure a numerical range (e.g., from 2342384 to 2342399) to discover other valid user identifiers proximate to Tom's ID. Set the step increment to 1.
- Start the Intruder attack.



[Figure 15- WebGoat]

Check the results table in Burp Intruder and look for differences in response length or status code, as these can show if a different user's profile was accessed. In this case, user ID 2342388 returned a response length of 467, while the others were 11977. When the response is viewed, it shows that the profile of another user, "Buffalo Bill," is now accessible.

This method highlights how, in real-world applications, attackers can exploit BAC by sending automated requests with different user or object IDs to access data that doesn't belong to them. Without proper server-side checks, the system may respond with sensitive information, such as personal profiles, financial records, or private documents. While WebGoat is used to demonstrate the concept in a safe environment, the same technique in real systems can lead to serious consequences, including data breaches, privacy violations, financial loss, and reputational damage.

Defense Methods (Shayan)

Server-Side Validation of Resource Access

- **Principle**: The paramount defense against IDOR and numerous BAC flaws involves the implementation of robust authorisation checks on the server-side for every request targeting a protected resource. The server must rigorously verify that the currently authenticated user possesses explicit permission to access the specific resource instance identified in the request (OWASP, 2013).
- **Technical Operation**:
    - Upon receiving a request for a resource (e.g., GET /api/users/2342388/profile), the server extracts the requested userId (2342388).
    - The server identifies the currently authenticated user via their session information (e.g., tom, whose internal identifier might be 101).
    - The server then consults its access control logic: "Is user tom (ID 101) authorised to view the profile of user ID 2342388?"
    - This validation might involve confirming if 101 == 2342388 (for users accessing their own profile) OR if user 101 possesses an administrative role that permits viewing other user profiles.
- **Mitigation Efficacy:** This directly thwarts the demonstrated IDOR attack. If Tom (ID 2342384) requests profile 2342388, the server-side validation (2342384 == 2342388) would fail, and access would be consequently denied (unless Tom has specific administrative privileges) (OWASP, 2013).

- **Pseudocode illustrating server-side IDOR validation**

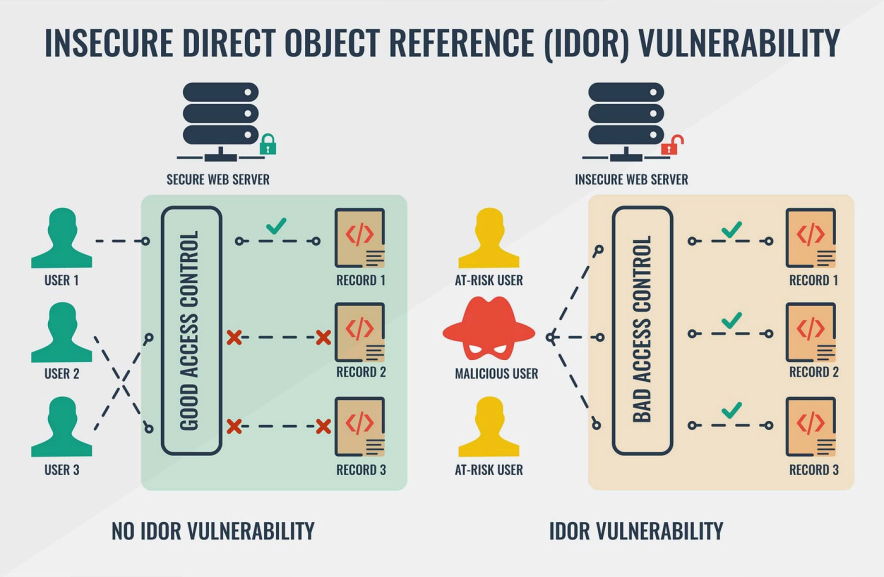
```
function getUserProfile(requestingUserSession, requestedProfileId):

  // Retrieve authenticated user from session

  currentUser = session.getCurrentAuthenticatedUser(requestingUserSession)



  // Verify if the authenticated user is requesting their own profile

  // OR if the authenticated user possesses administrative privileges

  if (currentUser.getInternalId() == requestedProfileId OR
currentUser.hasRole('ADMINISTRATOR')):

      // Proceed to fetch and return profile data for requestedProfileId
```

```
        profileData = database.fetchUserProfileById(requestedProfileId)

        return new Response(profileData, HTTP_STATUS_OK)

    else:

        // Log unauthorised access attempt

        securityLogger.warn("Unauthorised access attempt by user " + currentUser.getInternalId() + "
for profile " + requestedProfileId)

        return new Response("Access Denied", HTTP_STATUS_FORBIDDEN)
```

- **Implementation Considerations:** The validation rules should be used at every sensitive access point and endpoint. The system should be able to deal with many different roles and detailed permission levels.

Implementation of Indirect Object References for IDOR Prevention

- **Principle**: Avoid exposing direct, guessable, or enumerable internal identifiers (such as database primary keys) to the client. Instead, employ indirect references that are mapped to the actual internal identifiers on the server side. These indirect references should ideally be per-session, random, or unique to the user's context (OWASP, 2013).
- **Technical Operation:**
    - Upon user authentication, the application can generate a mapping of indirect references for objects the user is authorised to access. For instance, instead of Tom's profile being referenced via userId=2342384, it might be exposed as profileRef=aXbYcs123 in URLs or API calls for that specific session.
    - The server maintains this mapping (e.g., aXbYcs123 corresponds to internal ID 2342384).
    - When a request is received containing profileRef=aXbYcs123, the server consults the current user's session-specific map to resolve this indirect reference to the actual internal identifier.
- **Mitigation Efficacy:** As a result, it becomes much harder for an attacker to guess or list valid identifiers for other users' objects, because the exposed references are not in order. The indirect reference is only valid during the user's session, which makes it more secure (Legit Security, 2025).



(Dreamlab, 2020)

- **Implementation Considerations:** As a result, the server needs to handle the generation, management, and mapping of these indirect references. The references should be random and unique enough to be useful.

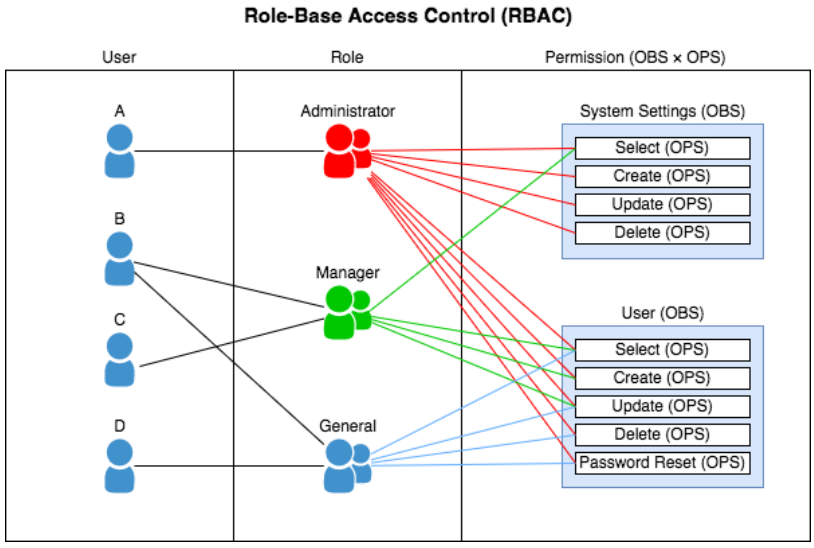
Adherence to the Principle of Least Privilege (PoLP)

- **Principle:** Users and system processes should be granted only the minimum set of permissions necessary to perform their designated tasks and functions. Access should be denied by default, with permissions explicitly granted only where essential (Least Privilege Violation | OWASP Foundation, n.d.).
- **Technical Operation:** During the design of roles and permissions, a thorough analysis must be conducted to determine the precise data and functionality each role legitimately requires. Overly broad or blanket permissions should be avoided. For example, a standard user should not possess permissions to view or modify data belonging to other users unless explicitly mandated by a specific application feature (e.g., collaborative document editing).

- **Mitigation Efficacy:** PoLP effectively reduces the potential attack surface. In the event of an account compromise, the potential damage is circumscribed by the minimal permissions assigned to that account.
- **Implementation Considerations:** Requires meticulous upfront analysis of application roles and functionalities. While potentially complex to manage in large-scale systems, it is a foundational security principle.

Application of Role-Based Access Control (RBAC)

- **Principle:** Assign permissions to defined roles rather than directly to individual user accounts. Users are then assigned to one or more of these roles (OWASP, n.d.).
- **Technical Operation:** Define distinct roles such as "StandardUser," "ContentEditor," and "SystemAdministrator." Each role is associated with a specific set of permissions (e.g., "StandardUser" can read their profile; "SystemAdministrator" can read/write any user's profile). Access control decisions are subsequently based on the roles assigned to the authenticated user.
- **Mitigation Efficacy:** RBAC simplifies permission management and promotes consistency. In the context of the demonstrated IDOR, if Tom is classified as a "StandardUser," RBAC would inherently restrict his access to his `userId` only.



Illustrative Role-Based Access Control Model.

(Sonoda, 2020)

- **Implementation Considerations**: Requires a clear and unambiguous definition of roles and their corresponding permissions. The system must reliably and consistently verify the user's roles for every sensitive operation.

Utilisation of Centralised Access Control Mechanisms

- **Principle:** Implement access control logic within a centralised, reusable, and thoroughly audited component or library, rather than distributing disparate checks throughout the application codebase (OWASP, 2021)
- **Technical Operation:** All requests for protected resources or functionalities are routed through this central authorisation module. This module typically takes the user's identity/roles and the requested resource/action as input and returns a definitive allow or deny decision.
- **Mitigation Efficacy:** This approach reduces the likelihood of developers inadvertently omitting access controls in new features or implementing them inconsistently or incorrectly. Centralisation also facilitates easier updates and maintenance of the access control logic (Team, 2025).
- **Implementation Considerations:** This can be a significant architectural decision. It requires careful design and robust implementation of the central authorisation service or library.

Discussion and Analysis

Technical Findings

The technical findings of this assignment reveal a successful exploitation of an IDOR vulnerability. The application failed to validate whether the authenticated user was authorised to access the profile linked to a supplied userId parameter. This lack of proper server-side authorisation checks allowed the user to retrieve another user’s data by modifying the ID in the request. The predictable and easily guessable format of the userId values made it possible to automate the attack using tools like Burp Intruder, which identified valid user IDs by analysing response differences. In a real-world scenario, this type of vulnerability could lead to unauthorised access to sensitive user data, including PII, potentially resulting in privacy breaches, identity theft, and financial fraud. If extended to functions that modify data, it could also allow attackers to alter or delete information belonging to other users. Although the demonstration was conducted in a simplified, educational environment using

WebGoat, the core issue, direct access to internal objects without authorisation checks, remains a serious and common weakness in real-world applications.

Justification

The proposed defense techniques offer a multi-layered, defense-in-depth approach to mitigate the demonstrated IDOR attack and BAC vulnerabilities more broadly.

- Server-side validation of resource access constitutes the most direct and effective countermeasure against the demonstrated IDOR. By ensuring the server explicitly verifies if the logged-in user is authorised for the specific resource instance requested, any attempt to access another user's data through ID manipulation would be effectively blocked. (OWASP, 2013).
- Indirect Object References significantly elevate the difficulty for attackers by obscuring the true internal identifiers. Even if an attacker could circumvent other checks, they would be unable to easily guess or enumerate the random or mapped references corresponding to other users' objects. This technique complements server-side validation by reducing the attack surface (Owasp, 2019).
- PoLP and RBAC are important security principles. Even if an attacker manages to access an account, the damage they can do is limited because that account has very few permissions. RBAC makes it easier to implement PoLP, as it is more reliable and less likely to have errors than random access checks. They help shape the way server-side validation is created.
- Centralised Access Control Mechanisms enhance the consistency and maintainability of authorisation logic. By establishing a single, well-tested point of enforcement, the risk of developers overlooking or incorrectly implementing access controls in disparate parts of the application is substantially reduced.

A layered security approach is essential, as no single defense is perfect. Even with server-side checks, an admin account with too many permissions can still cause problems. Therefore, combining PoLP, RBAC, strong server-side validation, and indirect references offers better protection. These defenses work best when they’re applied consistently and correctly. Ongoing monitoring and adapting to new threats are also key to keeping access control strong

References

OWASP. (2021). A01 Broken Access Control - OWASP Top 10:2021. Owasp.org; OWASP. https://owasp.org/Top10/A01_2021-Broken_Access_Control/

(OWASP, 2021)

Portswigger. (n.d.). Insecure direct object references (IDOR) | Web Security Academy. Portswigger.net. https://portswigger.net/web-security/access-control/idor

(Portswigger, n.d.)

WSTG - Latest | OWASP. (n.d.). Owasp.org. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/04-Testing_for_Insecure_Direct_Object_References

(WSTG - Latest | OWASP, n.d.)

OWASP. (2013). Session Management Cheat Sheet. Owasp.org. https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Akamai. (2024). What Are API Security Risks? | Akamai. Akamai. https://www.akamai.com/glossary/what-are-api-security-risks

What Is the Principle of Least Privilege? (n.d.). Palo Alto Networks. https://www.paloaltonetworks.com.au/cyberpedia/what-is-the-principle-of-least-privilege

Role-Based Access Control. (2022). Auth0 Docs. https://auth0.com/docs/manage-users/access-control/rbac

Owasp. (2019). Input Validation OWASP Cheat Sheet Series. Owasp.org; Owasp. https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

Dreamlab. (2020, March 18). Starbucks IDOR: How we prevented an information leak of 6 million Starbucks customers - Dreamlab Technologies. Dreamlab Technologies. https://dreamlab.net/en/blog/post/starbucks-idor-how-we-prevented-an-information-leak-of-6-million-starbucks-customers/

Sonoda, D. (2020, May 30). Role-based access control overview. Medium. https://dsonoda.medium.com/role-based-access-control-overview-257de64534c

Legit Security. (2025, April 22). What Are Insecure Direct Object References (IDOR)? Types and Prevention. Legitsecurity.com; Legit Security. https://www.legitsecurity.com/aspm-knowledge-base/insecure-direct-object-references/

Team, D. (2025, January 9). Centralized access control: efficient, secure, and scalable. Delinea; Delinea Inc. https://delinea.com/blog/centralized-access-management-explained

OWASP. (n.d.). Access Control for Software Security | OWASP Foundation. Owasp.org. https://owasp.org/www-community/Access_Control

Least Privilege Violation | OWASP Foundation. (n.d.). Owasp.org. https://owasp.org/www-community/vulnerabilities/Least_Privilege_Violation