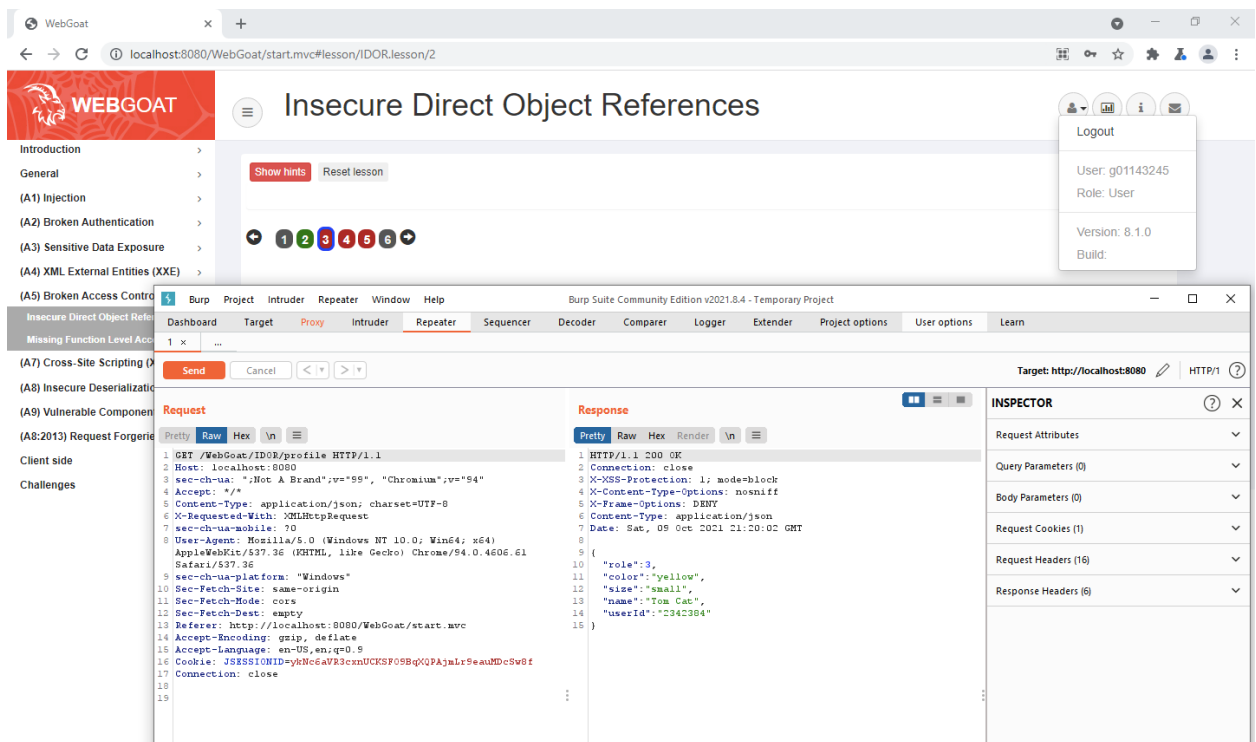
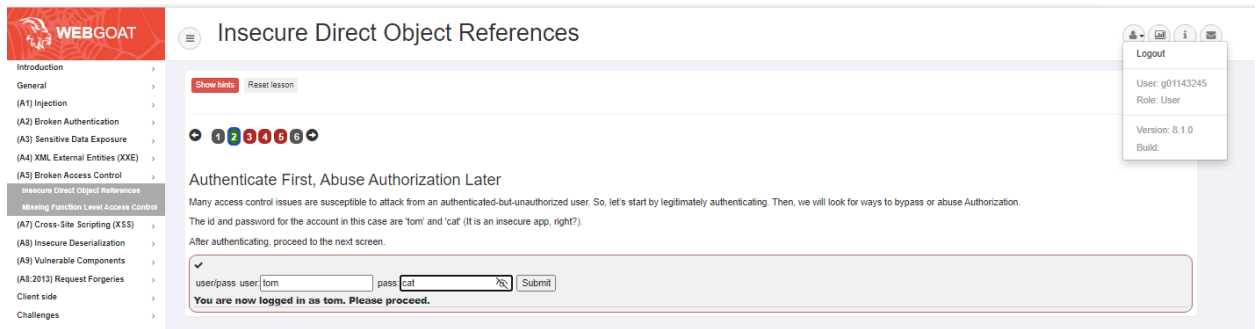


a.



WebGoat

localhost:8080/WebGoat/start.mvc#lesson/IDOR.lesson/2

## Insecure Direct Object References

Logout

User: g01143245  
Role: User

Version: 8.1.0  
Build:

Introduction >

General >

(A1) Injection >

(A2) Broken Authentication >

(A3) Sensitive Data Exposure >

(A4) XML External Entities (XXE) >

(A5) Broken Access Control >

**Insecure Direct Object References**

Missing Function Level Access Control >

(A7) Cross-Site Scripting (XSS) >

(A8) Insecure Deserialization >

(A9) Vulnerable Components >

(A8:2013) Request Forgeries >

Client side >

Challenges >

Show hints Reset lesson

1 2 3 4 5 6

### Observing Differences & Behaviors

A consistent principle from the offensive side of AppSec is to view differences from the raw response to what is visible. In other words (as you may have already noted in the client-side filtering lesson), there is often data in the raw response that doesn't show up on the screen/page. View the profile below and take note of the differences.

View Profile

name: Tom Cat  
color: yellow  
size: small

In the text input below, list the two attributes that are in the server's response, but don't show above in the profile.

role: userId Submit Diffs

WebGoat

localhost:8080/WebGoat/start.mvc#lesson/IDOR.lesson/2

## Insecure Direct Object References

Logout

User: g01143245  
Role: User

Version: 8.1.0  
Build:

Introduction >

General >

(A1) Injection >

(A2) Broken Authentication >

(A3) Sensitive Data Exposure >

(A4) XML External Entities (XXE) >

(A5) Broken Access Control >

**Insecure Direct Object References**

Missing Function Level Access Control >

(A7) Cross-Site Scripting (XSS) >

(A8) Insecure Deserialization >

(A9) Vulnerable Components >

(A8:2013) Request Forgeries >

Client side >

Challenges >

Show hints Reset lesson

1 2 3 4 5 6

### Observing Differences & Behaviors

A consistent principle from the offensive side of AppSec is to view differences from the raw response to what is visible. In other words (as you may have already noted in the client-side filtering lesson), there is often data in the raw response that doesn't show up on the screen/page. View the profile below and take note of the differences.

View Profile

name: Tom Cat  
color: yellow  
size: small

✓ In the text input below, list the two attributes that are in the server's response, but don't show above in the profile.

Submit Diffs

**Correct, the two attributes not displayed are userId & role. Keep those in mind**

b.

The screenshot shows the WebGoat application running in a browser at `localhost:8080/WebGoat/start.mvc#lesson/IDOR.lesson/3`. The lesson title is "Insecure Direct Object References". A sidebar on the left lists various security topics, with "Insecure Direct Object References" highlighted. A "Logout" button is visible in the top right corner. Overlaid on the application is the Burp Suite interface, showing a "Request" and "Response" tab. The "Request" tab displays a GET request to `/WebGoat/service/lessonoverview.mvc`. The "Response" tab shows a 200 OK status with a JSON body containing an "assignment" object. The "INSPECTOR" panel on the right shows the selected text `/IDOR/profile/(userId)` from the response body.

The screenshot shows the WebGoat application running in a browser at `localhost:8080/WebGoat/start.mvc#lesson/IDOR.lesson/3`. The lesson title is "Insecure Direct Object References". A sidebar on the left lists various security topics, with "Insecure Direct Object References" highlighted. A "Logout" button is visible in the top right corner. The main content area displays the lesson title and a "Show hints" button. Below the title, there is a section titled "Guessing & Predicting Patterns" with the subtitle "View Your Own Profile Another Way". The text explains that the application follows a RESTful pattern and that the user's session/authentication data won't tell them whose profile they want to view. A challenge box at the bottom asks the user to input an alternate path to the URL to view their own profile, starting with "WebGoat" (i.e. disregard "http://localhost:8080/"). The input field contains `Goat/IDOR/profile/2342384` and a "Submit" button.

WebGoat

## Insecure Direct Object References

Introduction >

General >

(A1) Injection >

(A2) Broken Authentication >

(A3) Sensitive Data Exposure >

(A4) XML External Entities (XXE) >

(A5) Broken Access Control >

**Insecure Direct Object References**

Missing Function Level Access Control >

(A7) Cross-Site Scripting (XSS) >

(A8) Insecure Deserialization >

(A9) Vulnerable Components >

(A8:2013) Request Forgeries >

Client side >

Challenges >

Show hints Reset lesson

1 2 3 4 5 6

### Guessing & Predicting Patterns

#### View Your Own Profile Another Way

The application we are working with seems to follow a RESTful pattern so far as the profile goes. Many apps have roles in which an elevated user may access content of another. In that case, just `/profile` won't work since the own user's session/authentication data won't tell us whose profile they want view. So, what do you think is a likely pattern to view your own profile explicitly using a direct object reference?

✓ Please input the alternate path to the Url to view your own profile. Please start with 'WebGoat' (i.e. disregard 'http://localhost:8080/')

WebGoat/ Submit

**Congratulations, you have used the alternate Url/route to view your own profile.**  
(role=3, color=yellow, size=small, name=Tom Cat, userId=2342384)

Logout

User: g01143245  
Role: User

Version: 8.1.0  
Build:

C.

WebGoat

## Insecure Direct Object References

Introduction >

General >

(A1) Injection >

(A2) Broken Authentication >

(A3) Sensitive Data Exposure >

(A4) XML External Entities (XXE) >

(A5) Broken Access Control >

**Insecure Direct Object References**

Missing Function Level Access Control >

(A7) Cross-Site Scripting (XSS) >

(A8) Insecure Deserialization >

(A9) Vulnerable Components >

(A8:2013) Request Forgeries >

Client side >

Challenges >

Show hints Reset lesson

1 2 3 4 5 6

### Playing with the Patterns

#### View Another Profile

View someone else's profile by using the alternate path you already used to view your own profile. Use the 'View Profile' button and intercept/modify the request to view another profile. Alternatively, you may also just be able to use a manual GET request with your browser.

View Profile

localhost:8080/WebGoat/IDOR/profile/2342388

```
{
  "lessonCompleted": true,
  "feedback": "Well done, you found someone else's profile",
  "output": "{role=3, color=brown, size=large, name=Buffalo Bill, userId=2342388}",
  "assignment": "IDORViewOtherProfile",
  "attemptsMade": true
}
```

Logout

User: g01143245  
Role: User

Version: 8.1.0  
Build:

The screenshot shows the WebGoat application interface with the lesson 'Insecure Direct Object References' selected. A Burp Suite window is overlaid, displaying a HTTP request and response. The request is a PUT to /WebGoat/IDOR/profile/2342388. The response is a 200 OK with a JSON body containing user information and a feedback message. A sidebar on the right shows a 'Logout' button and user details: User: g0t143245, Role: User, Version: 8.1.0, Build: .

d.

The screenshot shows the WebGoat application interface with the lesson 'Missing Function Level Access Control' selected. The lesson content includes a section 'Relying on Obscurity' and 'Finding Hidden Items'. A 'Your Mission' section is highlighted with a red box, containing a form with fields for 'Hidden Item 1' (Users) and 'Hidden Item 2' (Config), and a 'Submit' button. A sidebar on the right shows a 'Logout' button and user details: User: g0t143245, Role: User, Version: 8.1.0, Build: .

The screenshot shows the browser's developer tools with the 'Elements' tab selected. It displays the DOM structure of the 'Your Mission' section, showing a hidden menu item with a role of 'tabpanel'. The menu item contains two links: '/users' and '/config'. The 'Styles' panel on the right shows the default styles for the element.

**WEBGOAT** Missing Function Level Access Control

Introduction > General > (A1) Injection > (A2) Broken Authentication > (A3) Sensitive Data Exposure > (A4) XML External Entities (XXE) > (A5) Broken Access Control > **Insecure Direct Object References** > **Missing Function Level Access Control** > (A7) Cross-Site Scripting (XSS) > (A8) Insecure Deserialization > (A9) Vulnerable Components > (A8:2013) Request Forgeries > Client side > Challenges >

Show hints Reset lesson

1 2 3

## Relying on Obscurity

If you are relying on HTML, CSS or javascript to hide links that users don't normally access. It's a little older, but there was a case of a network router trying to protect (hide) admin functions with javascript in the UI <https://www.wired.com/2009/10/routers-still-vulnerable>

## Finding Hidden Items

There are usually hints to finding functionality the UI does not openly expose in ...

- HTML or javascript comments
- Commented out elements
- Items hidden via css controls/classes

## Your Mission

Find two menu items not visible in menu below that are or would be of interest to an attacker/malicious user and put the labels for those menu items (there are no links right now in the menus).

**Account**

My Profile

**Messages**

Log Out

✓

Hidden Item 1

Hidden Item 2

**Correct! And not hard to find are they?? One of these urls will be helpful in the next lab.**

e.

**WEBGOAT** Missing Function Level Access Control

Introduction > General > (A1) Injection > (A2) Broken Authentication > (A3) Sensitive Data Exposure > (A4) XML External Entities (XXE) > (A5) Broken Access Control > **Insecure Direct Object References** > **Missing Function Level Access Control** > (A7) Cross-Site Scripting (XSS) > (A8) Insecure Deserialization > (A9) Vulnerable Components > (A8:2013) Request Forgeries > Client side > Challenges >

Show hints Reset lesson

1 2 3

## Just Try It

As the previous page noted, sometimes apps rely on client controls. to control access (obscurity). If you can find items that don't have visible links, just try them, see what happens. Yes, it can be that simple!

## Gathering User Info

Often times, data dumps from vulnerabilities such as sql injection, but they can also come from poor or lacking access control.

It will likely take multiple steps and multiple attempts to get this one. Pay attention to the comments, leaked info. and you'll need to guess some. You may need to use another browser/account along the way. Start with the info. you already gathered (hidden menu items) to see if you can pull the list of users and then provide the 'Hash' for your own user account.

✓

Your Hash:

**Congrats! You really succeeded when you added the user.**

The screenshot displays the WebGoat application interface. On the left, a sidebar lists various lessons, with 'Missing Function Level Access Control' highlighted. The main content area shows the title 'Missing Function Level Access Control' and a 'Just Try It' section. Below this, there's a 'Gathering User Info' section with a form containing a 'Your Hash' field and a 'Submit' button. A message below the form reads 'Congrats! You really succeeded when you add...'. On the right, a user profile dropdown menu is visible, showing 'Logout', 'User: g0t1143245', 'Role: User', 'Version: 8.1.0', and 'Build:'. Overlaid on the bottom right is a 'Whitelabel Error Page' from the browser's developer tools. The error message states: 'This application has no explicit mapping for /error, so you are seeing this as a fallback. Sat Oct 09 18:38:39 EDT 2021. There was an unexpected error (type=Internal Server Error, status=500). Request processing failed; nested exception is org.thymeleaf.exceptions.TemplateInputException: An error happened during template rendering: unable to find a template for "classpath:/WEB-INF/templates/error.html", which might be in the package of a resource on the classpath.' The browser's network tab shows a 200 status code for a GET request to 'localhost:8080/WebGoat/users' with a JSON response containing user details.