Access control vulnerabilities and privilege escalation

## What is access control?

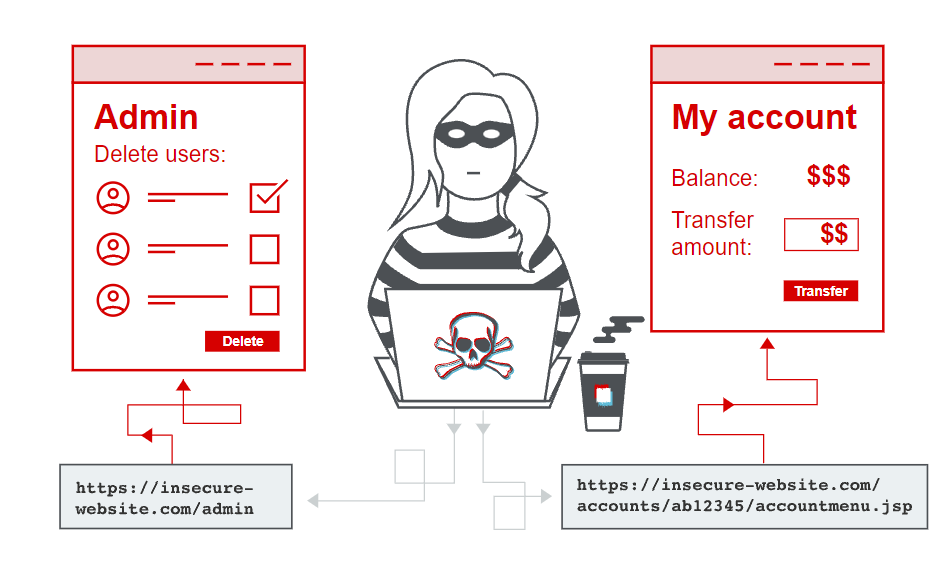
Access control (or authorization) is the application of constraints on who (or what) can perform attempted actions or access resources that they have requested. In the context of web applications, access control is dependent on authentication and session management:

* **Authentication** identifies the user and confirms that they are who they say they are.
* **Session management** identifies which subsequent HTTP requests are being made by that same user.
* **Access control** determines whether the user is allowed to carry out the action that they are attempting to perform.

Broken access controls are a commonly encountered and often critical security vulnerability. Design and management of access controls is a complex and dynamic problem that applies business, organizational, and legal constraints to a technical implementation. Access control design decisions have to be made by humans, not technology, and the potential for errors is high.

From a user perspective, access controls can be divided into the following categories:

* [Vertical access controls](https://portswigger.net/web-security/access-control#vertical-access-controls)
* [Horizontal access controls](https://portswigger.net/web-security/access-control#horizontal-access-controls)
* [Context-dependent access controls](https://portswigger.net/web-security/access-control#context-dependent-access-controls)



# Access control security models

In this section we explain what [access control](https://portswigger.net/web-security/access-control) security models are and we discuss the most commonly encountered models.

## What are access control security models?

An access control security model is a formally defined definition of a set of access control rules that is independent of technology or implementation platform. Access control security models are implemented within operating systems, networks, database management systems and back office, application and web server software. Various access control security models have been devised over the years to match access control policies to business or organizational rules and changes in technology.

## Programmatic access control

With programmatic access control, a matrix of user privileges is stored in a database or similar and [access controls](https://portswigger.net/web-security/access-control) are applied programmatically with reference to this matrix. This approach to access control can include roles or groups or individual users, collections or workflows of processes and can be highly granular.

## Discretionary access control (DAC)

With discretionary access control, access to resources or functions is constrained based upon users or named groups of users. Owners of resources or functions have the ability to assign or delegate access permissions to users. This model is highly granular with access rights defined to an individual resource or function and user. Consequently the model can become very complex to design and manage.

## Mandatory access control (MAC)

Mandatory access control is a centrally controlled system of access control in which access to some object (a file or other resource) by a subject is constrained. Significantly, unlike DAC the users and owners of resources have no capability to delegate or modify access rights for their resources. This model is often associated with military clearance-based systems.

## Role-based access control (RBAC)

With role-based access control, named roles are defined to which access privileges are assigned. Users are then assigned to single or multiple roles. RBAC provides enhanced management over other access control models and if properly designed sufficient granularity to provide manageable access control in complex applications. For example, the purchase clerk might be defined as a role with access permissions for a subset of purchase ledger functionality and resources. As employees leave or join an organization then access control management is simplified to defining or revoking membership of the purchases clerk role.

RBAC is most effective when there are sufficient roles to properly invoke access controls but not so many as to make the model excessively complex and unwieldy to manage.

# Access control vulnerabilities and privilege escalation

In this section, we will discuss what access control security is, describe privilege escalation and the types of vulnerabilities that can arise with access control, and summarize how to prevent these vulnerabilities.

## What is access control?

Access control (or authorization) is the application of constraints on who (or what) can perform attempted actions or access resources that they have requested. In the context of web applications, access control is dependent on authentication and session management:

* **Authentication** identifies the user and confirms that they are who they say they are.
* **Session management** identifies which subsequent HTTP requests are being made by that same user.
* **Access control** determines whether the user is allowed to carry out the action that they are attempting to perform.

Broken access controls are a commonly encountered and often critical security vulnerability. Design and management of access controls is a complex and dynamic problem that applies business, organizational, and legal constraints to a technical implementation. Access control design decisions have to be made by humans, not technology, and the potential for errors is high.

From a user perspective, access controls can be divided into the following categories:

* [Vertical access controls](https://portswigger.net/web-security/access-control#vertical-access-controls)
* [Horizontal access controls](https://portswigger.net/web-security/access-control#horizontal-access-controls)
* [Context-dependent access controls](https://portswigger.net/web-security/access-control#context-dependent-access-controls)

#### Read more

[Access control security models](https://portswigger.net/web-security/access-control/security-models)

### Vertical access controls

Vertical access controls are mechanisms that restrict access to sensitive functionality that is not available to other types of users.

With vertical access controls, different types of users have access to different application functions. For example, an administrator might be able to modify or delete any user's account, while an ordinary user has no access to these actions. Vertical access controls can be more fine-grained implementations of security models designed to enforce business policies such as separation of duties and least privilege.

### Horizontal access controls

Horizontal access controls are mechanisms that restrict access to resources to the users who are specifically allowed to access those resources.

With horizontal access controls, different users have access to a subset of resources of the same type. For example, a banking application will allow a user to view transactions and make payments from their own accounts, but not the accounts of any other user.

### Context-dependent access controls

Context-dependent access controls restrict access to functionality and resources based upon the state of the application or the user's interaction with it.

Context-dependent access controls prevent a user performing actions in the wrong order. For example, a retail website might prevent users from modifying the contents of their shopping cart after they have made payment.

## Examples of broken access controls

Broken access control vulnerabilities exist when a user can in fact access some resource or perform some action that they are not supposed to be able to access.

### Vertical privilege escalation

If a user can gain access to functionality that they are not permitted to access then this is vertical privilege escalation. For example, if a non-administrative user can in fact gain access to an admin page where they can delete user accounts, then this is vertical privilege escalation.

#### Unprotected functionality

At its most basic, vertical privilege escalation arises where an application does not enforce any protection over sensitive functionality. For example, administrative functions might be linked from an administrator's welcome page but not from a user's welcome page. However, a user might simply be able to access the administrative functions by browsing directly to the relevant admin URL.

For example, a website might host sensitive functionality at the following URL:

https://insecure-website.com/admin

This might in fact be accessible by any user, not only administrative users who have a link to the functionality in their user interface. In some cases, the administrative URL might be disclosed in other locations, such as the robots.txt file:

https://insecure-website.com/robots.txt

Even if the URL isn't disclosed anywhere, an attacker may be able to use a wordlist to brute-force the location of the sensitive functionality.

# Lab: Unprotected admin functionality

This lab has an unprotected admin panel.

Solve the lab by deleting the user carlos.

#### Solution

1. Go to the lab and view robots.txt by appending /robots.txt to the lab URL. Notice that the Disallow line discloses the path to the admin panel.
2. In the URL bar, replace /robots.txt with /administrator-panel to load the admin panel.
3. Delete carlos.

# Access control vulnerabilities and privilege escalation

In this section, we will discuss what access control security is, describe privilege escalation and the types of vulnerabilities that can arise with access control, and summarize how to prevent these vulnerabilities.

## What is access control?

Access control (or authorization) is the application of constraints on who (or what) can perform attempted actions or access resources that they have requested. In the context of web applications, access control is dependent on authentication and session management:

* **Authentication** identifies the user and confirms that they are who they say they are.
* **Session management** identifies which subsequent HTTP requests are being made by that same user.
* **Access control** determines whether the user is allowed to carry out the action that they are attempting to perform.

Broken access controls are a commonly encountered and often critical security vulnerability. Design and management of access controls is a complex and dynamic problem that applies business, organizational, and legal constraints to a technical implementation. Access control design decisions have to be made by humans, not technology, and the potential for errors is high.

From a user perspective, access controls can be divided into the following categories:

* [Vertical access controls](https://portswigger.net/web-security/access-control#vertical-access-controls)
* [Horizontal access controls](https://portswigger.net/web-security/access-control#horizontal-access-controls)
* [Context-dependent access controls](https://portswigger.net/web-security/access-control#context-dependent-access-controls)

#### Read more

[Access control security models](https://portswigger.net/web-security/access-control/security-models)

### Vertical access controls

Vertical access controls are mechanisms that restrict access to sensitive functionality that is not available to other types of users.

With vertical access controls, different types of users have access to different application functions. For example, an administrator might be able to modify or delete any user's account, while an ordinary user has no access to these actions. Vertical access controls can be more fine-grained implementations of security models designed to enforce business policies such as separation of duties and least privilege.

### Horizontal access controls

Horizontal access controls are mechanisms that restrict access to resources to the users who are specifically allowed to access those resources.

With horizontal access controls, different users have access to a subset of resources of the same type. For example, a banking application will allow a user to view transactions and make payments from their own accounts, but not the accounts of any other user.

### Context-dependent access controls

Context-dependent access controls restrict access to functionality and resources based upon the state of the application or the user's interaction with it.

Context-dependent access controls prevent a user performing actions in the wrong order. For example, a retail website might prevent users from modifying the contents of their shopping cart after they have made payment.

## Examples of broken access controls

Broken access control vulnerabilities exist when a user can in fact access some resource or perform some action that they are not supposed to be able to access.

### Vertical privilege escalation

If a user can gain access to functionality that they are not permitted to access then this is vertical privilege escalation. For example, if a non-administrative user can in fact gain access to an admin page where they can delete user accounts, then this is vertical privilege escalation.

#### Unprotected functionality

At its most basic, vertical privilege escalation arises where an application does not enforce any protection over sensitive functionality. For example, administrative functions might be linked from an administrator's welcome page but not from a user's welcome page. However, a user might simply be able to access the administrative functions by browsing directly to the relevant admin URL.

For example, a website might host sensitive functionality at the following URL:

https://insecure-website.com/admin

This might in fact be accessible by any user, not only administrative users who have a link to the functionality in their user interface. In some cases, the administrative URL might be disclosed in other locations, such as the robots.txt file:

https://insecure-website.com/robots.txt

Even if the URL isn't disclosed anywhere, an attacker may be able to use a wordlist to brute-force the location of the sensitive functionality.

**LAB**[Unprotected admin functionality](https://portswigger.net/web-security/access-control/lab-unprotected-admin-functionality)Solved

In some cases, sensitive functionality is not robustly protected but is concealed by giving it a less predictable URL: so called security by obscurity. Merely hiding sensitive functionality does not provide effective access control since users might still discover the obfuscated URL in various ways.

For example, consider an application that hosts administrative functions at the following URL:

https://insecure-website.com/administrator-panel-yb556

This might not be directly guessable by an attacker. However, the application might still leak the URL to users. For example, the URL might be disclosed in JavaScript that constructs the user interface based on the user's role:

<script>  
var isAdmin = false;  
if (isAdmin) {  
  ...  
  var adminPanelTag = document.createElement('a');  
  adminPanelTag.setAttribute('https://insecure-website.com/administrator-panel-yb556');  
  adminPanelTag.innerText = 'Admin panel';  
  ...  
}  
</script>

This script adds a link to the user's UI if they are an admin user. However, the script containing the URL is visible to all users regardless of their role.

# Lab: Unprotected admin functionality with unpredictable URL

This lab has an unprotected admin panel. It's located at an unpredictable location, but the location is disclosed somewhere in the application.

Solve the lab by accessing the admin panel, and using it to delete the user carlos.

#### Solution

1. Review the lab home page's source using Burp Suite or your web browser's developer tools.
2. Observe that it contains some JavaScript that discloses the URL of the admin panel.
3. Load the admin panel and delete carlos.

#### Parameter-based access control methods

Some applications determine the user's access rights or role at login, and then store this information in a user-controllable location, such as a hidden field, cookie, or preset query string parameter. The application makes subsequent access control decisions based on the submitted value. For example:

https://insecure-website.com/login/home.jsp?admin=true  
https://insecure-website.com/login/home.jsp?role=1

This approach is fundamentally insecure because a user can simply modify the value and gain access to functionality to which they are not authorized, such as administrative functions.

# Lab: User role controlled by request parameter

This lab has an admin panel at /admin, which identifies administrators using a forgeable cookie.

Solve the lab by accessing the admin panel and using it to delete the user carlos.

You can log in to your own account using the following credentials: wiener:peter

#### Solution

1. Browse to /admin and observe that you can't access the admin panel.
2. Browse to the login page.
3. In Burp Proxy, turn interception on and enable response interception.
4. Complete and submit the login page, and forward the resulting request in Burp.
5. Observe that the response sets the cookie Admin=false. Change it to Admin=true.
6. Load the admin panel and delete carlos.

# Lab: User role can be modified in user profile

This lab has an admin panel at /admin, which identifies administrators using a forgeable cookie.

Solve the lab by accessing the admin panel and using it to delete the user carlos.

You can log in to your own account using the following credentials: wiener:peter

#### Solution

1. Browse to /admin and observe that you can't access the admin panel.
2. Browse to the login page.
3. In Burp Proxy, turn interception on and enable response interception.
4. Complete and submit the login page, and forward the resulting request in Burp.
5. Observe that the response sets the cookie Admin=false. Change it to Admin=true.
6. Load the admin panel and delete carlos.

Lab: User role can be modified in user profile

This lab has an admin panel at /admin. It's only accessible to logged-in users with a roleid of 2.

Solve the lab by accessing the admin panel and using it to delete the user carlos.

You can log in to your own account using the following credentials: wiener:peter

#### Solution

1. Log in using the supplied credentials and access your account page.
2. Use the provided feature to update the email address associated with your account.
3. Observe that the response contains your role ID.
4. Send the email submission request to Burp Repeater, add "roleid":2 into the JSON in the request body, and resend it.
5. Observe that the response shows your roleid has changed to 2.
6. Browse to /admin and delete carlos.

#### roken access control resulting from platform misconfiguration

Some applications enforce access controls at the platform layer by restricting access to specific URLs and HTTP methods based on the user's role. For example an application might configure rules like the following:

DENY: POST, /admin/deleteUser, managers

This rule denies access to the POST method on the URL /admin/deleteUser, for users in the managers group. Various things can go wrong in this situation, leading to access control bypasses.

Some application frameworks support various non-standard HTTP headers that can be used to override the URL in the original request, such as X-Original-URL and X-Rewrite-URL. If a web site uses rigorous front-end controls to restrict access based on URL, but the application allows the URL to be overridden via a request header, then it might be possible to bypass the access controls using a request like the following:

POST / HTTP/1.1  
X-Original-URL: /admin/deleteUser

Lab: URL-based access control can be circumvented

This website has an unauthenticated admin panel at /admin, but a front-end system has been configured to block external access to that path. However, the back-end application is built on a framework that supports the X-Original-URL header.

To solve the lab, access the admin panel and delete the user carlos.

#### Solution

1. Try to load /admin and observe that you get blocked. Notice that the response is very plain, suggesting it may originate from a front-end system.
2. Send the request to Burp Repeater. Change the URL in the request line to / and add the HTTP header X-Original-URL: /invalid. Observe that the application returns a "not found" response. This indicates that the back-end system is processing the URL from the X-Original-URL header.
3. Change the value of the X-Original-URL header to /admin. Observe that you can now access the admin page.
4. To delete the user carlos, add ?username=carlos to the real query string, and change the X-Original-URL path to /admin/delete.

Lab: Method-based access control can be circumvented

This lab implements [access controls](https://portswigger.net/web-security/access-control) based partly on the HTTP method of requests. You can familiarize yourself with the admin panel by logging in using the credentials administrator:admin.

To solve the lab, log in using the credentials wiener:peter and exploit the flawed access controls to promote yourself to become an administrator.

#### Solution

1. Log in using the admin credentials.
2. Browse to the admin panel, promote carlos, and send the HTTP request to Burp Repeater.
3. Open a private/incognito browser window, and log in with the non-admin credentials.
4. Attempt to re-promote carlos with the non-admin user by copying that user's session cookie into the existing Burp Repeater request, and observe that the response says "Unauthorized".
5. Change the method from POST to POSTX and observe that the response changes to "missing parameter".
6. Convert the request to use the GET method by right-clicking and selecting "Change request method".
7. Change the username parameter to your username and resend the request.

### Horizontal privilege escalation

Horizontal privilege escalation arises when a user is able to gain access to resources belonging to another user, instead of their own resources of that type. For example, if an employee should only be able to access their own employment and payroll records, but can in fact also access the records of other employees, then this is horizontal privilege escalation.

Horizontal privilege escalation attacks may use similar types of exploit methods to vertical privilege escalation. For example, a user might ordinarily access their own account page using a URL like the following:

https://insecure-website.com/myaccount?id=123

Now, if an attacker modifies the id parameter value to that of another user, then the attacker might gain access to another user's account page, with associated data and functions.

Lab: User ID controlled by request parameter

This lab has a horizontal privilege escalation vulnerability on the user account page.

To solve the lab, obtain the API key for the user carlos and submit it as the solution.

You can log in to your own account using the following credentials: wiener:peter

#### Solution

1. Log in using the supplied credentials and go to your account page.
2. Note that the URL contains your username in the "id" parameter.
3. Send the request to Burp Repeater.
4. Change the "id" parameter to carlos.
5. Retrieve and submit the API key for carlos.

In some applications, the exploitable parameter does not have a predictable value. For example, instead of an incrementing number, an application might use globally unique identifiers (GUIDs) to identify users. Here, an attacker might be unable to guess or predict the identifier for another user. However, the GUIDs belonging to other users might be disclosed elsewhere in the application where users are referenced, such as user messages or reviews.

Lab: User ID controlled by request parameter, with unpredictable user IDs

This lab has a horizontal privilege escalation vulnerability on the user account page, but identifies users with GUIDs.

To solve the lab, find the GUID for carlos, then submit his API key as the solution.

You can log in to your own account using the following credentials: wiener:peter

#### Solution

1. Find a blog post by carlos.
2. Click on carlos and observe that the URL contains his user ID. Make a note of this ID.
3. Log in using the supplied credentials and access your account page.
4. Change the "id" parameter to the saved user ID.
5. Retrieve and submit the API key.

In some cases, an application does detect when the user is not permitted to access the resource, and returns a redirect to the login page. However, the response containing the redirect might still include some sensitive data belonging to the targeted user, so the attack is still successful.

Lab: User ID controlled by request parameter with data leakage in redirect

This lab contains an [access control](https://portswigger.net/web-security/access-control) vulnerability where sensitive information is leaked in the body of a redirect response.

To solve the lab, obtain the API key for the user carlos and submit it as the solution.

You can log in to your own account using the following credentials: wiener:peter

#### Solution

1. Log in using the supplied credentials and access your account page.
2. Send the request to Burp Repeater.
3. Change the "id" parameter to carlos.
4. Observe that although the response is now redirecting you to the home page, it has a body containing the API key belonging to carlos.
5. Submit the API key.

### Horizontal to vertical privilege escalation

Often, a horizontal privilege escalation attack can be turned into a vertical privilege escalation, by compromising a more privileged user. For example, a horizontal escalation might allow an attacker to reset or capture the password belonging to another user. If the attacker targets an administrative user and compromises their account, then they can gain administrative access and so perform vertical privilege escalation.

For example, an attacker might be able to gain access to another user's account page using the parameter tampering technique already described for horizontal privilege escalation:

https://insecure-website.com/myaccount?id=456

If the target user is an application administrator, then the attacker will gain access to an administrative account page. This page might disclose the administrator's password or provide a means of changing it, or might provide direct access to privileged functionality.

Lab: User ID controlled by request parameter with password disclosure

This lab has user account page that contains the current user's existing password, prefilled in a masked input.

To solve the lab, retrieve the administrator's password, then use it to delete carlos.

You can log in to your own account using the following credentials: wiener:peter

#### Solution

1. Log in using the supplied credentials and access the user account page.
2. Change the "id" parameter in the URL to "administrator".
3. View the response in Burp and observe that it contains the administrator's password.
4. Log in to the administrator account and delete carlos.

### [Insecure direct object references](https://portswigger.net/web-security/access-control/idor)

Insecure direct object references (IDOR) are a subcategory of access control vulnerabilities. IDOR arises when an application uses user-supplied input to access objects directly and an attacker can modify the input to obtain unauthorized access. It was popularized by its appearance in the OWASP 2007 Top Ten although it is just one example of many implementation mistakes that can lead to access controls being circumvented.

Lab: Insecure direct object references

This lab stores user chat logs directly on the server's file system, and retrieves them using static URLs.

Solve the lab by finding the password for the user carlos, and logging into their account.

#### Solution

1. Select the "Live chat" tab.
2. Send a message and then select "View transcript".
3. Review the URL and observe that the transcripts are text files assigned a filename containing an incrementing number.
4. Change the filename to 1.txt and review the text. Notice a password within the chat transcript.
5. Return to the main lab page and log in using the stolen credentials.

# Insecure direct object references (IDOR)

In this section, we will explain what insecure direct object references (IDOR) are and describe some common vulnerabilities.

## What are insecure direct object references (IDOR)?

Insecure direct object references (IDOR) are a type of [access control](https://portswigger.net/web-security/access-control) vulnerability that arises when an application uses user-supplied input to access objects directly. The term IDOR was popularized by its appearance in the OWASP 2007 Top Ten. However, it is just one example of many access control implementation mistakes that can lead to [access controls](https://portswigger.net/web-security/access-control) being circumvented. IDOR vulnerabilities are most commonly associated with horizontal privilege escalation, but they can also arise in relation to vertical privilege escalation.

## IDOR examples

There are many examples of access control vulnerabilities where user-controlled parameter values are used to access resources or functions directly.

### IDOR vulnerability with direct reference to database objects

Consider a website that uses the following URL to access the customer account page, by retrieving information from the back-end database:

https://insecure-website.com/customer\_account?customer\_number=132355

Here, the customer number is used directly as a record index in queries that are performed on the back-end database. If no other controls are in place, an attacker can simply modify the customer\_number value, bypassing access controls to view the records of other customers. This is an example of an IDOR vulnerability leading to horizontal privilege escalation.

An attacker might be able to perform horizontal and vertical privilege escalation by altering the user to one with additional privileges while bypassing access controls. Other possibilities include exploiting password leakage or modifying parameters once the attacker has landed in the user's accounts page, for example.

### Access control vulnerabilities in multi-step processes

Many web sites implement important functions over a series of steps. This is often done when a variety of inputs or options need to be captured, or when the user needs to review and confirm details before the action is performed. For example, administrative function to update user details might involve the following steps:

1. Load form containing details for a specific user.
2. Submit changes.
3. Review the changes and confirm.

Sometimes, a web site will implement rigorous access controls over some of these steps, but ignore others. For example, suppose access controls are correctly applied to the first and second steps, but not to the third step. Effectively, the web site assumes that a user will only reach step 3 if they have already completed the first steps, which are properly controlled. Here, an attacker can gain unauthorized access to the function by skipping the first two steps and directly submitting the request for the third step with the required parameters.

Lab: Multi-step process with no access control on one step

This lab has an admin panel with a flawed multi-step process for changing a user's role. You can familiarize yourself with the admin panel by logging in using the credentials administrator:admin.

To solve the lab, log in using the credentials wiener:peter and exploit the flawed [access controls](https://portswigger.net/web-security/access-control) to promote yourself to become an administrator.

#### Solution

1. Log in using the admin credentials.
2. Browse to the admin panel, promote carlos, and send the confirmation HTTP request to Burp Repeater.
3. Open a private/incognito browser window, and log in with the non-admin credentials.
4. Copy the non-admin user's session cookie into the existing Repeater request, change the username to yours, and replay it.

### Referer-based access control

Some websites base access controls on the Referer header submitted in the HTTP request. The Referer header is generally added to requests by browsers to indicate the page from which a request was initiated.

**For example, suppose an application robustly enforces access control over the main administrative page at /admin, but for sub-pages such as /admin/deleteUser only inspects the Referer header. If the Referer header contains the main /admin URL, then the request is allowed.**

In this situation, since the Referer header can be fully controlled by an attacker, they can forge direct requests to sensitive sub-pages, supplying the required Referer header, and so gain unauthorized access.

Lab: Referer-based access control

This lab controls access to certain admin functionality based on the Referer header. You can familiarize yourself with the admin panel by logging in using the credentials administrator:admin.

To solve the lab, log in using the credentials wiener:peter and exploit the flawed [access controls](https://portswigger.net/web-security/access-control) to promote yourself to become an administrator.

#### Solution

1. Log in using the admin credentials.
2. Browse to the admin panel, promote carlos, and send the HTTP request to Burp Repeater.
3. Open a private/incognito browser window, and log in with the non-admin credentials.
4. Browse to /admin-roles?username=carlos&action=upgrade and observe that the request is treated as unauthorized due to the absent Referer header.
5. Copy the non-admin user's session cookie into the existing Burp Repeater request, change the username to yours, and replay it.

### Location-based access control

Some web sites enforce access controls over resources based on the user's geographical location. This can apply, for example, to banking applications or media services where state legislation or business restrictions apply. These access controls can often be circumvented by the use of web proxies, VPNs, or manipulation of client-side geolocation mechanisms.

## How to prevent access control vulnerabilities

Access control vulnerabilities can generally be prevented by taking a defense-in-depth approach and applying the following principles:

* Never rely on obfuscation alone for access control.
* Unless a resource is intended to be publicly accessible, deny access by default.
* Wherever possible, use a single application-wide mechanism for enforcing access controls.
* At the code level, make it mandatory for developers to declare the access that is allowed for each resource, and deny access by default.
* Thoroughly audit and test access controls to ensure they are working as designed.