Access Control

⋮ 27/07/2022

Jul 26, 2022

7 min read
BAC Web-Notes

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.

References

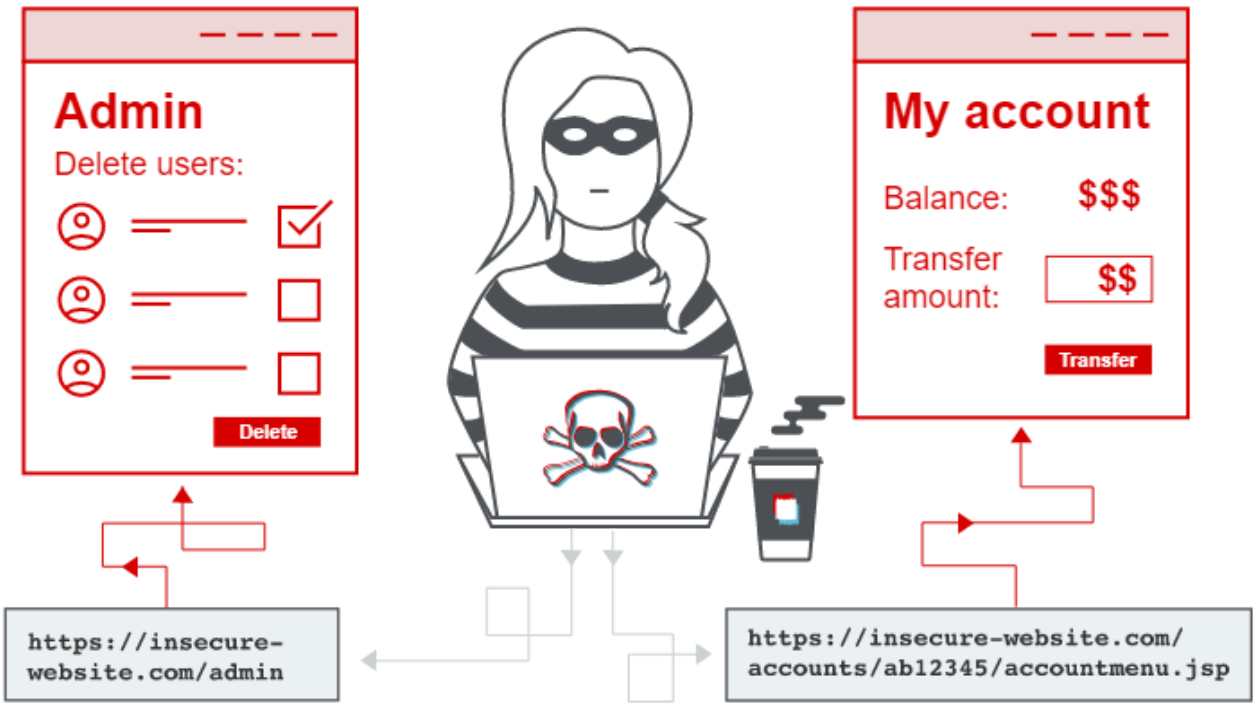
Hacksplaining: Web Security for Developers

Mind-Maps/Access Control Vulnerabilities

All labs - Web Security Academy

What is access control?

- **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.



Vertical access controls

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

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.

Horizontal access controls

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

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 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 happens when a user can perform actions he mustn't do.

Vertical privilege escalation

Access to function that are no permited to access to.

EX: `User` can Access to `admin` panel and delete any user account

Unprotected functionality

For example, a website might host sensitive functionality at the following URL: `https://insecure-website.com/admin`

Even if it's not publicly accessed to any user, but may some attackers access to it form : `https://insecure-website.com/robots.txt`

Lab: Unprotected admin functionality

In some cases, sensitive functionality is not robustly protected but is concealed by giving it a less predictable URL:

For example, consider an application that hosts administrative functions at the following URL: `https://insecure-website.com/administrator-panel-yb556`

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>
```

Lab: Unprotected admin functionality with unpredictable URL

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

Lab: User role controlled by request parameter

Lab: User role can be modified in user profile

Broken access control resulting from platform misconfiguration

For example an application might configure rules like the following: `DENY: POST, /admin/deleteUser, managers`

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

If an attacker can use the `GET`(or another) method to perform actions on a restricted URL, then they can circumvent the access control that is implemented at the platform layer.

Lab: Method-based access control can be circumvented

Horizontal privilege escalation

Horizontal privilege escalation arises when a `user` is able to gain access to resources belonging to `another user`, 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`.**

For example, a user might ordinarily access their own account page using a URL like the following: `[https://insecure-website.com/myaccount?id=123](https://insecure-website.com/myaccount?id=123)` if the attacker could modify the id parameter !!

Lab: User ID controlled by request parameter

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.

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

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

---

Horizontal to vertical privilege escalation

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.

Attacker might gain horizonal access to another user account and if he an admin he will make a vertical privillage esclation

Lab: User ID controlled by request parameter with password disclosure

---

Insecure direct object references (IDOR)

Its an Access Control Vulnerability that arises when an application uses user-supplied input to access objects directly.

IDOR vulnerabilities are most commonly associated with horizontal privilege escalation but can arise also with vertical one.

IDOR examples

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 If the customer number not secure enough the attacker can modify it and break the access control

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

For example, a website might save `chat message transcripts` to disk using an incrementing filename, and allow users to retrieve these by visiting a URL like the following:

`https://insecure-website.com/static/12144.txt`

In this situation, an attacker can simply **`modify the filename to retrieve a transcript created by another user`** and potentially obtain user credentials and other sensitive data.

Lab: Insecure direct object references

Access control vulnerabilities in multi-step processes

Many web sites implement `important functions` over a series of steps.

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.

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

Referer-based access control

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

Prevention

- 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.