Broken Access Control

White Hats: Team Members

Banks Cargill - cargillb@oregonstate.edu

Frederick Eley - eleyf@oregonstate.edu

Timothy Glew - glewt@oregonstate.edu

Description:

Another web application security risk listed by the OWASP Foundation is broken access control. Access control is also commonly referred to as authorization. Once a user is authenticated and logged into a web application, restrictions on what data and resources that user may access should be implemented. Failure to secure access can lead to the exposure of sensitive data which may be viewed, modified, or deleted. An attacker may even be able to take control of an entire system if they gain administrative rights. Access controls are divided into three different categories: vertical, horizontal, and context-dependent, each having its own vulnerabilities.

Vertical Access Controls

Vertical access controls allow different types of users to access different data and functions in an application. This is often seen in the difference between administrators and regular users. An administrator typically can modify or delete accounts. If a regular user were to gain unauthorized access to an admin page they would not only be able to view and/or download all users' data, they would also be able to modify or delete the accounts making them inaccessible to the original users. Vertical privilege escalation often occurs when administrative data and functions are at a specific URL that contain no authorization checks. For example, an administrative panel might be at the URL http://foo-bar.com/admin. A developer may only have a link to this page visible when an administrator logs in and presumes that it is secure because only authenticated admins will see this link. However, the page is insecure, anyone with knowledge of this URL can now access the administrative panel. The URL is visible on-screen and may be seen by shoulder surfers. It could also be stored in the browser history or proxy logs. Furthermore, administrators might even write down, bookmark, or email this URL. Finally, the location of an administrative URL may be discovered by reading a web application's robots.txt file or by brute forcing a URL with a wordlist.

Horizontal Access Controls

Horizontal access controls allow users to access data and resources of the same type. For example, an online store will allow a user to see the items in their shopping cart but not those of another user. Horizontal privilege escalation occurs when a user can gain access to data and resources belonging to another user, most commonly by exploiting the URL parameters. For example, a user may log in to an account page with the URL http://foo-bar.com.com/account?id=123. In an application with no access controls, simply changing the id parameter could allow access to another user's account. Specific accounts may be targeted particularly when the URL parameters are not integers but user IDs such as an account for Jane

Doe at http://foo-bar.com.com/account?id=doej. The problem here originates from user IDs being accessible through means outside the web application such as email messages.

Context-Dependent Access Controls

Context-dependent access controls restrict access to functionality and resources according to the state of the application or the user's interaction. This prevents a user from performing actions out of order. For example, an online store prevents users from modifying the contents of their shopping cart after they have made payment. Much like the other types of access vulnerabilities, context-dependent privilege escalation can occur by URL exploitation but in a slightly different way. An application may store access information in a hidden field, cookie, or preset query string parameter. For example, at checkout a shipping invoice may be filled when a user reaches the URL, http://foo-bar.com/shipping.jsp?paid=true, indicating that the payment process has been cleared. Without access controls, a user with knowledge of this URL could bypass the payment page and go directly to the shipping phase.

In summary, the best ways to prevent access control vulnerabilities are to always assume that URL locations will be found, manually or by automation, and access should be granted only through proper authorization. Public access should be denied by default with exceptions being created if the resource is intended to be publicly accessible. Always remember to log access control failures and alert administrators on repeated failures and create limits to failed access attempts and your site should be well on its way to being secure against broken access control.

How to Attack our Site

Vertical Privilege Escalation

Our web application was built with one account type, that of a general user. Since there are no administrative accounts or rights to exploit, our application is not vulnerable to this type of attack.

Horizontal Privilege Escalation

To exploit this vulnerability login to the insecure web app with your created account. Once at the home page type into the browser's URL:

```
https://osu-capstone-project-insecure.herokuapp.com/tasks/1
```

You will be taken to one of the lists of a default account.

Next, enter:

```
https://osu-capstone-project-insecure.herokuapp.com/tasks/11
```

You will have access to another list which belongs to a default account.

Finally, enter:

```
https://osu-capstone-project-insecure.herokuapp.com/tasks/21
```

You will again have access to a list which belongs to another user. Even though these lists belong to other users, you can edit or delete any of the listed tasks. You can add new tasks to lists as well. Keep in mind that this also exposes your own lists to others that may want to exploit this vulnerability. Being that the URL variable enabling access to the lists is an integer it would be possible for an attacker to write a script to quickly identify all exposed lists.

Context-Dependent Privilege Escalation

Our web application was built as a simple to list that allows a user to create, edit, or delete content as they please. Since there is no context dependent logic flow in this design, our application is not vulnerable to this type of attack.

How to Defend our Site

Horizontal Privilege Escalation

To prevent horizontal privilege escalation, when a call to view a list is made an authorization process is undertaken before the list is rendered. The list data returned from the database query contains the user id of that list as its first parameter and it is compared to the user id of the logged in user (see line 529 below). If the authorization process fails, the list is not rendered but instead the failed attempt to access the list is logged (line 532) and the user is redirected to see an error that they are being denied access to view a list that does not belong to them (line 533).

```
518  def tasks(list_id):
519      """
520      Route for the tasks page of a user's list where all of the tasks of a to do list are shown
521      """
522      db_connection = connect_to_database()  # connect to db
523
524      # check if requested list belongs to the user
525      cursor = db_connection.cursor()
526      cursor.callproc('getListUserId', [list_id, ])
527      rtn = cursor.fetchall()
528      cursor.close()
529      if rtn[0][0] != current_user.id:
530          print(rtn)
531          db_connection.close() # close connection before returning
532          webapp.logger.WARNING("Invalid access to view list. userid: %s", current_user.id)
533          return redirect(url_for('invalid_access'))
534
```

References

https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control

https://portswigger.net/web-security/access-control

Stuttard, Dafydd, and Marcus Pinto. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Second Edition*. John Wiley & Sons, Inc., 2011.