

# Authorization Testing



---

Instructor: C. Pu (Ph.D., Assistant Professor)

*puc@marshall.edu*



# Introduction

---

- Authorization is the concept of allowing access to resources only to those permitted to use them.
- Testing for Authorization means *understanding how the authorization process works*, and using that information to circumvent the authorization mechanism.
- Authorization is a process that comes after a successful authentication, so the tester will verify this point after he holds valid credentials, associated with a well-defined set of roles and privileges.
- During this kind of assessment, it should be verified if it is possible to bypass the authorization schema, find a path traversal vulnerability, or find ways to escalate the privileges assigned to the tester.



# Testing for Bypassing Authorization Schema: Summary

---

- This kind of test focuses on verifying how the authorization schema has been implemented for each role or privilege to get access to reserved functions and resources.
- For every specific role the tester holds during the assessment, for every function and request that the application executes during the post-authentication phase, it is necessary to verify:
  - Is it possible to access that resource even if the user is not authenticated?
  - Is it possible to access that resource after the log-out?
  - Is it possible to access functions and resources that should be accessible to a user that holds a different role or privilege?



# Testing for Bypassing Authorization Schema: Summary

---

- Try to access the application as an administrative user and track all the administrative functions.
  - Is it possible to access administrative functions also if the tester is logged as a user with standard privileges?
  - Is it possible to use these administrative functions as a user with a different role and for whom that action should be denied?

# Testing for Bypassing Authorization Schema: How to Test

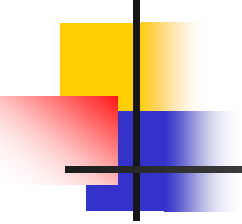
- Testing for access to administrative functions
- For example, suppose that the 'AddUser.jsp' function is part of the administrative menu of the application, and it is possible to access it by requesting the following URL:

```
https://www.example.com/admin/addUser.jsp
```

- Then, the following HTTP request is generated when calling the AddUser function:

```
POST /admin/addUser.jsp HTTP/1.1
Host: www.example.com
[other HTTP headers]

userID=fakeuser&role=3&group=grp001
```



# Testing for Bypassing Authorization Schema: How to Test

---

- Testing for access to resources assigned to a different role
- For example, analyze an application that uses a shared directory to store temporary PDF files for different users.
  - Suppose that documentABC.pdf should be accessible only by the user test1 with roleA.
  - Verify if user test2 with roleB can access that resource.



# Testing for Privilege Escalation: Summary

---

- Privilege escalation occurs when a user gets access to more resources or functionality than they are normally allowed, and such elevation or changes should have been prevented by the application.
- This is usually caused by a flaw in the application.
- The result is that the application performs actions with more privileges than those intended by the developer or system administrator.

# Testing for Privilege Escalation: Summary



---

- The degree of escalation depends on what privileges the attacker is authorized to possess, and what privileges can be obtained in a successful exploit.
- For example, a programming error that allows a user to gain extra privilege after successful authentication limits the degree of escalation, because the user is already authorized to hold some privilege.
- Likewise, a remote attacker gaining superuser privilege without any authentication presents a greater degree of escalation.





# Testing for Privilege Escalation: Summary

---

- Usually, people refer to *vertical escalation* when it is possible to access resources granted to more privileged accounts (e.g., acquiring administrative privileges for the application), and to *horizontal escalation* when it is possible to access resources granted to a similarly configured account (e.g., in an online banking application, accessing information related to a different user).



# Testing for Privilege Escalation: How to Test

---

- Testing for role/privilege manipulation
  - In every portion of the application where a user can create information in the database (e.g., making a payment, adding a contact, or sending a message), can receive information (statement of account, order details, etc.), or delete information (drop users, messages, etc.), it is necessary to record that functionality.
  - The tester should try to access such functions as another user in order to verify if it is possible to access a function that should not be permitted by the user's role/privilege (but might be permitted as another user).

# Testing for Privilege Escalation: How to Test

- For example
  - The following HTTP POST allows the user that belongs to grp001 to access order #0001:

```
POST /admin/addUser.jsp HTTP/1.1
Host: www.example.com
[other HTTP headers]

userID=fakeuser&role=3&group=grp001
```

- Verify if a user that does not belong to grp001 can modify the value of the parameters 'groupID' and 'orderID' to gain access to that privileged data.

# Testing for Privilege Escalation: How to Test

- For example

- The following server's answer shows a hidden field in the HTML returned to the user after a successful authentication.

- hidden field:

- [https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5\\_input\\_type\\_hidden](https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5_input_type_hidden)

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/6.0
Date: Wed, 1 Apr 2006 13:51:20 GMT
Set-Cookie: USER=aW78ryrGrTWs4MnOd32Fs51yDqp; path=/;
domain=www.example.com
Set-Cookie: SESSION=k+KmKeHXTgDi1J5fT7Zz; path=/;
domain= www.example.com
Cache-Control: no-cache
Pragma: No-cache
Content-length: 247
Content-Type: text/html
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Connection: close
```

```
<form name="autoriz" method="POST" action="visual.jsp">
<input type="hidden" name="profile" value="SysAdmin">
<body onload="document.forms.autoriz.submit()">
</td>
</tr>
```

# Testing for Insecure Direct Object References: Summary

- Insecure Direct Object References occur when an application provides direct access to objects based on user-supplied input.
- As a result of this vulnerability attackers can bypass authorization and access resources in the system directly, for example database records or files.
- Insecure Direct Object References allow attackers to bypass authorization and access resources directly by modifying the value of a parameter used to directly point to an object.
- Such resources can be database entries belonging to other users, files in the system, and more.
- This is caused by the fact that the application takes user supplied input and uses it to retrieve an object without performing sufficient authorization checks.



# Testing for Insecure Direct Object References: How to Test

---

- To test for this vulnerability the tester first needs to map out all locations in the application where user input is used to reference objects directly.
- For example, locations where user input is used to access a database row, a file, application pages and more.
- Next the tester should modify the value of the parameter used to reference objects and assess whether it is possible to retrieve objects belonging to other users or otherwise bypass authorization.



# Testing for Insecure Direct Object References: How to Test

---

- The best way to test for direct object references would be by having at least two (often more) users to cover different owned objects and functions.
- For example two users each having access to different objects (such as purchase information, private messages, etc.), and (if relevant) users with different privileges (for example administrator users) to see whether there are direct references to application functionality.
- By having multiple users the tester saves valuable testing time in guessing different object names as he can attempt to access objects that belong to the other user.

# Testing for Insecure Direct Object References: How to Test

- The value of a parameter is used directly to retrieve a database record

- Sample request:

<http://foo.bar/somepage?invoice=12345>

- In this case, the value of the invoice parameter is used as an index in an invoices table in the database.
- The application takes the value of this parameter and uses it in a query to the database.
- The application then returns the invoice information to the user.
- Since the value of invoice goes directly into the query, by modifying the value of the parameter it is possible to retrieve any invoice object, regardless of the user to whom the invoice belongs.



# Testing for Insecure Direct Object References: How to Test

- The value of a parameter is used directly to perform an operation in the system

- Sample request:

```
http://foo.bar/changepassword?user=someuser
```

- In this case, the value of the user parameter is used to tell the application for which user it should change the password.
- In many cases this step will be a part of a wizard, or a multi-step operation.
- In the first step the application will get a request stating for which user's password is to be changed, and in the next step the user will provide a new password (without asking for the current one).

# Testing for Insecure Direct Object References: How to Test

- The value of a parameter is used directly to perform an operation in the system

- Sample request:

```
http://foo.bar/changepassword?user=someuser
```

- The user parameter is used to directly reference the object of the user for whom the password change operation will be performed.
- To test for this case the tester should attempt to provide a different test username than the one currently logged in, and check whether it is possible to modify the password of another user.

# Testing for Insecure Direct Object References: How to Test

- The value of a parameter is used directly to retrieve a file system resource
  - Sample request:

```
http://foo.bar/showImage?img=img00011
```

- In this case, the value of the file parameter is used to tell the application what file the user intends to retrieve.
- By providing the name or identifier of a different file (for example file=image00012.jpg) the attacker will be able to retrieve objects belonging to other users.

# Testing for Insecure Direct Object References: How to Test

- The value of a parameter is used directly to access application functionality

- Sample request:

<http://foo.bar/accessPage?menuitem=12>

- In this case, the value of the menuitem parameter is used to tell the application which menu item (and therefore which application functionality) the user is attempting to access.
- Assume the user is supposed to be restricted and therefore has links available only to access to menu items 1, 2 and 3. By modifying the value of menuitem parameter it is possible to bypass authorization and access additional application functionality.
- To test for this case the tester identifies a location where application functionality is determined by reference to a menu item, maps the values of menu items the given test user can access, and then attempts other menu items.