# Identity Management Testing Report

## OTG-IDENT-001: Test Role Definitions

### Test Objective

This test aims to verify that the application correctly enforces role-based access control (RBAC) and that users can only access functionalities and data appropriate to their assigned roles. It ensures that lower-privileged users cannot access or manipulate resources intended for higher-privileged users (e.g., administrators).

### Target Endpoint

The target endpoints for this test are various pages within the DVWA application that are expected to have different access levels based on user roles. Examples include:

- `/security.php` (Expected: Administrator access only)
- `/setup.php` (Expected: Administrator access only)
- `/index.php` (Expected: Accessible to all logged-in users)
- Any other pages or functionalities that should be restricted based on user roles.

### Methodology

The methodology involves attempting to access restricted resources with lower-privileged accounts or unauthenticated sessions. The Python script simulates user logins with different credentials and then attempts to navigate to pages or perform actions that are not permitted for the logged-in user's role. The responses (HTTP status codes, page content) are then analyzed to determine if access control is properly enforced.

**Important Python Snippet:**

```
import requests
```

```
# Example function to log in and get a session
def login(session, username, password):
    login_url = "http://localhost/dvwa/login.php"
    payload = {
        "username": username,
        "password": password,
        "Login": "Login"
    }
    response = session.post(login_url, data=payload)
    return response

# Example function to test access to a restricted page
def test_access(session, url):
    response = session.get(url)
    return response.status_code, response.text

# Main test logic (simplified)
if __name__ == "__main__":
    dvwa_base_url = "http://localhost/dvwa/"

    # Test with a low-privileged user
    with requests.Session() as low_priv_session:
        login(low_priv_session, "user", "password")
        admin_page_url = dvwa_base_url + "security.php"
        status_code, content = test_access(low_priv_session,
admin_page_url)
        print(f"Low-privileged user access to {admin_page_url}:
Status {status_code}")
        # Expected: Redirect to login or error page, not actual
admin content

    # Test with an administrator user
    with requests.Session() as admin_session:
        login(admin_session, "admin", "password")
        admin_page_url = dvwa_base_url + "security.php"
        status_code, content = test_access(admin_session,
admin_page_url)
        print(f"Admin user access to {admin_page_url}: Status
{status_code}")
```

```
        # Expected: 200 OK and actual admin content
```

**Specific Library or Tools:**

- `requests`: A popular Python library for making HTTP requests. Essential for interacting with web applications.
- `BeautifulSoup` (optional): For parsing HTML content to verify specific elements or messages on the page.
- Web proxy tools like Burp Suite or OWASP ZAP can be used manually to observe and manipulate requests during testing.

## Step to Reproduce

---

1. **Setup DVWA:** Ensure DVWA is running and accessible (e.g., at `http://localhost/dvwa/`). Set the security level to "low" initially for easier observation, then test with higher levels.
2. **Identify Restricted Pages:** Manually identify pages or functionalities within DVWA that should only be accessible by specific roles (e.g., `/security.php` and `/setup.php` for administrators).
3. **Login as Low-Privileged User:** Use the Python script or manually log in to DVWA with a low-privileged account (e.g., username: `user`, password: `password`).
4. **Attempt Unauthorized Access:** While logged in as the low-privileged user, attempt to navigate directly to the identified restricted pages (e.g., `http://localhost/dvwa/security.php`).
5. **Observe Response:**
    - **Expected Result (Secure):** The application should redirect the user to the login page, display an "Access Denied" message, or show a generic error page, preventing access to the restricted content. The HTTP status code should typically be 302 (Found/Redirect) or 403 (Forbidden).
    - **Actual Result (Vulnerable):** If the low-privileged user gains access to the restricted content, it indicates a role definition vulnerability.
6. **Login as Administrator:** Log in to DVWA with an administrator account (e.g., username: `admin`, password: `password`).
7. **Verify Authorized Access:** Attempt to navigate to the same restricted pages.
8. **Observe Response:**
    - **Expected Result:** The administrator should be able to access and view the content of these pages without any issues (HTTP status code 200 OK).

9. **Log Evidence:** Record the HTTP requests, responses (status codes, headers, and relevant body content), and any error messages or unexpected behaviors observed during the test.

## Log Evidence

```
# Example Log Evidence for OTG-IDENT-001

--- Test Case: OTG-IDENT-001 - Test Role Definitions ---
Target URL: http://localhost/dvwa/security.php

--- Attempt with Low-Privileged User (user:password) ---
Request: GET http://localhost/dvwa/security.php
Response Status Code: 302 Found (or 403 Forbidden)
Response Headers:
    Location: http://localhost/dvwa/login.php (if redirected)
Response Body (snippet, if applicable):
    <h1>Login</h1> (if redirected to login page)
    <p>Access Denied</p> (if forbidden)
Observation: User was correctly denied access to the security
settings page.

--- Attempt with Administrator User (admin:password) ---
Request: GET http://localhost/dvwa/security.php
Response Status Code: 200 OK
Response Headers:
    Content-Type: text/html; charset=utf-8
Response Body (snippet):
    <h1>Security Level</h1>
    <p>Choose your security level:</p>
Observation: Administrator was correctly granted access to the
security settings page.

--- Conclusion ---
Role definitions appear to be correctly enforced for the tested
pages.
```

# OTG-IDENT-002: Test User Registration Process

## Test Objective

This test aims to evaluate the security of the user registration process. It focuses on identifying vulnerabilities such as insecure default settings, lack of input validation, weak password policy enforcement, and potential for account creation abuse (e.g., creating multiple accounts, bypassing CAPTCHA).

## Target Endpoint

For DVWA, there is typically no direct user registration page by default. New users are usually created by an administrator. If a registration page were present, the target endpoint would be:

- `http://localhost/dvwa/register.php` (Hypothetical registration page)
- Any API endpoints involved in user creation.

**Note:** As DVWA does not have a public registration page, this test often involves assessing the account provisioning process (OTG-IDENT-003) or the administrator's user creation functionality.

## Methodology

If a registration page exists, the methodology involves:

- Attempting to register with various invalid inputs (e.g., too short/long usernames/passwords, special characters, SQL injection payloads, XSS payloads).
- Testing for weak password policy enforcement (e.g., registering with "password123").
- Attempting to bypass CAPTCHA or other anti-automation mechanisms.
- Registering multiple accounts to check for rate limiting or account creation limits.
- Checking for default or guessable usernames/passwords upon registration.

**Important Python Snippet:**

```
import requests
```

```
# Hypothetical registration function
def register_user(session, username, password,
confirm_password):
    register_url = "http://localhost/dvwa/register.php" #
Hypothetical
    payload = {
        "username": username,
        "password": password,
        "confirm_password": confirm_password,
        "Register": "Register"
    }
    response = session.post(register_url, data=payload)
    return response.status_code, response.text

# Main test logic (simplified for a hypothetical registration)
if __name__ == "__main__":
    dvwa_base_url = "http://localhost/dvwa/"

    # Test with weak password
    with requests.Session() as s:
        status_code, content = register_user(s, "testuser1",
"123", "123")
        print(f"Registration with weak password: Status
{status_code}")
        # Expected: Error message about weak password

    # Test with SQL injection in username
    with requests.Session() as s:
        status_code, content = register_user(s, "' OR 1=1--",
"password", "password")
        print(f"Registration with SQLi username: Status
{status_code}")
        # Expected: Input validation error or generic error
```

**Specific Library or Tools:**

- `requests`: For sending registration requests.
- `BeautifulSoup`: To parse registration form errors or success messages.

- Burp Suite/OWASP ZAP: For intercepting and modifying registration requests, fuzzing inputs.

## Step to Reproduce

1. **Identify Registration Endpoint:** If DVWA had a public registration page, locate its URL (e.g., `http://localhost/dvwa/register.php`).
2. **Test Input Validation:**
    - Attempt to register with empty fields.
    - Attempt to register with excessively long usernames/passwords.
    - Attempt to register with special characters or known attack payloads (e.g., `<script>alert('XSS')</script>` in username).
    - Observe error messages and server responses.
3. **Test Password Policy:**
    - Attempt to register with very simple passwords (e.g., "123", "password").
    - Attempt to register with passwords that do not meet common complexity requirements (e.g., no uppercase, no numbers, no special characters).
    - Observe if the application enforces a strong password policy.
4. **Test Account Enumeration during Registration:** If the registration form indicates whether a username is already taken, this could lead to enumeration.
5. **Test Rate Limiting/Abuse:** Attempt to register multiple accounts rapidly to see if rate limiting or other anti-abuse mechanisms are in place.
6. **Log Evidence:** Record all requests, responses, and observations.

## Log Evidence

```
# Example Log Evidence for OTG-IDENT-002 (Hypothetical)

--- Test Case: OTG-IDENT-002 - Test User Registration Process -
--
Target URL: http://localhost/dvwa/register.php (Hypothetical)

--- Attempt: Registration with weak password "123" ---
Request: POST http://localhost/dvwa/register.php
Payload:
username=newuser&password=123&confirm_password=123&Register=Reg
ister
```

```
Response Status Code: 200 OK (or 400 Bad Request)
Response Body (snippet):
    <p class="error">Password is too short or too simple.</p>
Observation: Application correctly rejected weak password.

--- Attempt: Registration with SQL Injection in username ---
Request: POST http://localhost/dvwa/register.php
Payload: username=' OR 1=1--
&password=securepass&confirm_password=securepass&Register=Regis
ter
Response Status Code: 200 OK
Response Body (snippet):
    <p class="error">Invalid characters in username.</p>
Observation: Application performed input validation on
username.

--- Conclusion ---
(Based on observations)
If DVWA had a registration page, the tests would indicate its
robustness.
```

# OTG-IDENT-003: Test Account Provisioning Process

## Test Objective

This test evaluates the security of how user accounts are created, modified, and deleted within the application, typically by an administrator. It aims to identify vulnerabilities in the provisioning workflow, such as insecure default permissions, lack of proper authorization checks, or information leakage during account management.

## Target Endpoint

In DVWA, account provisioning is primarily handled through the administrator interface. The key endpoint is:

- `http://localhost/dvwa/setup.php` (Administrator setup page where users can be created/reset)
- Any underlying scripts or API calls initiated from this page for user management.

## Methodology

The methodology involves interacting with the account provisioning functionalities as an administrator and also attempting to access or manipulate these functionalities as a non-administrator. It includes:

- Verifying that only authorized users (administrators) can provision accounts.
- Checking for default or weak initial passwords for newly provisioned accounts.
- Testing for proper logging of account provisioning actions.
- Attempting to bypass authorization checks to create/modify/delete accounts without administrative privileges.

**Important Python Snippet:**

```
import requests

# Function to log in (re-used from OTG-IDENT-001)
def login(session, username, password):
    login_url = "http://localhost/dvwa/login.php"
    payload = {
        "username": username,
        "password": password,
        "Login": "Login"
    }
    response = session.post(login_url, data=payload)
    return response

# Function to attempt to reset/create DB (simulates
provisioning)
def attempt_db_reset(session):
    setup_url = "http://localhost/dvwa/setup.php"
    # This payload simulates clicking the "Create / Reset
Database" button
    # which also creates default users.
    payload = {
```

```
        "create_db": "Create / Reset Database"
    }
    response = session.post(setup_url, data=payload)
    return response.status_code, response.text

# Main test logic
if __name__ == "__main__":
    dvwa_base_url = "http://localhost/dvwa/"

    # Attempt provisioning as a low-privileged user
    with requests.Session() as low_priv_session:
        login(low_priv_session, "user", "password")
        status_code, content =
attempt_db_reset(low_priv_session)
        print(f"Low-privileged user attempt to provision:
Status {status_code}")
        # Expected: Redirect to login or error, not successful
provisioning

    # Attempt provisioning as an administrator
    with requests.Session() as admin_session:
        login(admin_session, "admin", "password")
        status_code, content = attempt_db_reset(admin_session)
        print(f"Admin user attempt to provision: Status
{status_code}")
        # Expected: 200 OK and success message (database
reset/users created)
```

**Specific Library or Tools:**

- `requests`: For sending HTTP requests to the setup page.
- Burp Suite/OWASP ZAP: For intercepting and modifying requests to test authorization bypasses.

## Step to Reproduce

1. **Setup DVWA:** Ensure DVWA is running.

2. **Identify Provisioning Functionality:** Navigate to `http://localhost/dvwa/setup.php` as an administrator. Observe the "Create / Reset Database" button, which also provisions default user accounts.
3. **Test Unauthorized Access to Provisioning:**
    - Log out of DVWA, or open a new browser/session.
    - Attempt to directly access `http://localhost/dvwa/setup.php` without logging in, or after logging in as a low-privileged user (e.g., `user:password`).
    - Attempt to send a POST request to `setup.php` with the `create_db` parameter, simulating the button click, while unauthenticated or as a low-privileged user.
    - **Observe Response:**
        - **Expected Result (Secure):** Access should be denied, or the action should fail due to insufficient privileges.
        - **Actual Result (Vulnerable):** If the database is reset or users are created/modified without proper authorization, it indicates a vulnerability.
4. **Test Authorized Provisioning:**
    - Log in as an administrator (`admin:password`).
    - Navigate to `http://localhost/dvwa/setup.php` and click "Create / Reset Database".
    - **Observe Response:**
        - **Expected Result:** The database should be reset, and default users should be provisioned successfully.
5. **Log Evidence:** Record all requests, responses, and observations, noting any instances where unauthorized provisioning was possible or where default passwords were weak.

## Log Evidence

```
# Example Log Evidence for OTG-IDENT-003

--- Test Case: OTG-IDENT-003 - Test Account Provisioning
Process ---
Target URL: http://localhost/dvwa/setup.php

--- Attempt with Unauthenticated Session ---
Request: POST http://localhost/dvwa/setup.php
Payload: create_db=Create / Reset Database
```

```
Response Status Code: 302 Found
Response Headers:
    Location: http://localhost/dvwa/login.php
Observation: Unauthenticated access to provisioning
functionality was correctly denied.

--- Attempt with Low-Privileged User (user:password) ---
Request: POST http://localhost/dvwa/setup.php
Payload: create_db=Create / Reset Database
Response Status Code: 302 Found
Response Headers:
    Location: http://localhost/dvwa/login.php
Observation: Low-privileged user access to provisioning
functionality was correctly denied.

--- Attempt with Administrator User (admin:password) ---
Request: POST http://localhost/dvwa/setup.php
Payload: create_db=Create / Reset Database
Response Status Code: 200 OK
Response Body (snippet):
    <p>Database has been created.</p>
Observation: Administrator successfully initiated database
reset and user provisioning.

--- Conclusion ---
Account provisioning functionality appears to be adequately
protected by authorization checks in DVWA.
```

# OTG-IDENT-004: Testing for Account Enumeration and Guessable User Account

## Test Objective

This test aims to identify if an attacker can determine valid usernames within the application, typically by observing differences in error messages or response times during

login, password reset, or registration attempts. It also checks for easily guessable default or common usernames.

## Target Endpoint

The primary target endpoint for account enumeration in DVWA is the login page:

- `http://localhost/dvwa/login.php`
- Any password reset or account recovery pages (if present).

## Methodology

The methodology involves sending login requests with a mix of valid and invalid usernames, combined with arbitrary passwords. The script then analyzes the application's responses (error messages, HTTP status codes, response times) to identify any discernible differences that reveal whether a username exists. For guessable accounts, common usernames (e.g., "admin", "test", "user") are attempted.

**Important Python Snippet:**

```
import requests
import time

def attempt_login(session, username, password):
    login_url = "http://localhost/dvwa/login.php"
    payload = {
        "username": username,
        "password": password,
        "Login": "Login"
    }
    start_time = time.time()
    response = session.post(login_url, data=payload)
    end_time = time.time()
    return response.status_code, response.text, (end_time -
start_time)

# Main test logic
if __name__ == "__main__":
```

```
    dvwa_base_url = "http://localhost/dvwa/"

    valid_username = "admin"
    invalid_username = "nonexistentuser123"
    dummy_password = "anypassword"

    with requests.Session() as s:
        print("--- Testing for Account Enumeration ---")

        # Test with a valid username
        status_code_valid, content_valid, time_valid =
attempt_login(s, valid_username, dummy_password)
        print(f"Attempt with valid username '{valid_username}':
Status {status_code_valid}, Time {time_valid:.4f}s")
        print(f"Response snippet:
{content_valid[content_valid.find('

'):content_valid.find('

')+4]}")

        # Test with an invalid username
        status_code_invalid, content_invalid, time_invalid =
attempt_login(s, invalid_username, dummy_password)
        print(f"Attempt with invalid username
'{invalid_username}': Status {status_code_invalid}, Time
{time_invalid:.4f}s")
        print(f"Response snippet:
{content_invalid[content_invalid.find('

'):content_invalid.find('

')+4]}")

        # Analyze differences
        if "Username and/or password incorrect." in
content_valid and "Username and/or password incorrect." in
content_invalid:
            print("Observation: Generic error message for both
valid and invalid usernames. No direct enumeration via error
messages.")
```

```
        elif "Username exists but password incorrect." in
content_valid and "Username does not exist." in
content_invalid:
            print("Observation: Different error messages for
valid vs. invalid usernames. Account enumeration possible.")
        else:
            print("Observation: Further analysis needed for
error message differences.")

        print("
--- Testing for Guessable User Accounts ---")
        guessable_users = ["admin", "user", "test", "root",
"guest"]
        for user in guessable_users:
            status_code, content, _ = attempt_login(s, user,
dummy_password)
            if "Username and/or password incorrect." not in
content: # Or other success indicator
                print(f"Guessable user '{user}' might exist (or
login was successful with dummy password).")
```

**Specific Library or Tools:**

- `requests`: For sending login requests and analyzing responses.
- `time`: For measuring response times (though often less reliable than error messages).
- Burp Suite/OWASP ZAP Intruder/Fuzzer: Excellent for automating enumeration attempts and analyzing differences in responses.

## Step to Reproduce

1. **Setup DVWA:** Ensure DVWA is running.
2. **Access Login Page:** Navigate to `http://localhost/dvwa/login.php`.
3. **Test Error Message Differences:**
    - Attempt to log in with a known valid username (e.g., `admin`) and an incorrect password. Note the exact error message displayed.
    - Attempt to log in with a known invalid username (e.g., `nonexistentuser123`) and any password. Note the exact error message

displayed.
    - **Observe Response:**
        - **Expected Result (Secure):** Both attempts should yield the exact same generic error message (e.g., "Username and/or password incorrect.").
        - **Actual Result (Vulnerable):** If the error messages differ (e.g., "Invalid password for user 'admin'" vs. "Username 'nonexistentuser123' not found"), account enumeration is possible.
4. **Test Response Time Differences (Less Reliable):**
    - Repeat the above steps, but also measure the response time for each login attempt.
    - **Observe Response:** Significant and consistent differences in response times (e.g., valid usernames taking consistently longer) can sometimes indicate enumeration, but this is less reliable than error messages.
5. **Test for Guessable User Accounts:**
    - Attempt to log in with common or default usernames (e.g., `admin`, `user`, `test`, `root`, `guest`) and an arbitrary password.
    - Observe if any of these attempts yield a different error message or behavior, suggesting the username exists.
6. **Log Evidence:** Record the usernames used, the exact error messages received, HTTP status codes, and response times for each attempt.

## Log Evidence

```
# Example Log Evidence for OTG-IDENT-004

--- Test Case: OTG-IDENT-004 - Testing for Account Enumeration
---
Target URL: http://localhost/dvwa/login.php

--- Attempt 1: Valid Username (admin), Invalid Password
(wrongpass) ---
Request: POST http://localhost/dvwa/login.php
Payload: username=admin&password=wrongpass&Login=Login
Response Status Code: 200 OK
Response Body (snippet):
    <p class="error">Username and/or password incorrect.</p>
Response Time: 0.1234s
```

```
Observation: Standard error message for incorrect credentials.

--- Attempt 2: Invalid Username (nonexistentuser), Any Password
(anypass) ---
Request: POST http://localhost/dvwa/login.php
Payload: username=nonexistentuser&password=anypass&Login=Login
Response Status Code: 200 OK
Response Body (snippet):
    <p class="error">Username and/or password incorrect.</p>
Response Time: 0.1250s
Observation: Same error message as for valid username. No
direct enumeration via error messages.

--- Test Case: OTG-IDENT-004 - Testing for Guessable User
Accounts ---

--- Attempt 1: Guessable Username (user), Invalid Password
(wrongpass) ---
Request: POST http://localhost/dvwa/login.php
Payload: username=user&password=wrongpass&Login=Login
Response Status Code: 200 OK
Response Body (snippet):
    <p class="error">Username and/or password incorrect.</p>
Observation: 'user' account exists in DVWA, but error message
is generic.

--- Conclusion ---
DVWA's login page provides a generic error message for both
valid and invalid usernames, which mitigates direct account
enumeration via error messages. However, common usernames like
'admin' and 'user' are default and easily guessable.
```

## OTG-IDENT-005: Testing for Weak or unenforced username policy

### Test Objective

This test aims to identify if the application enforces a strong username policy. It checks for vulnerabilities such as allowing overly simple, common, or sensitive information as usernames, or permitting special characters that could lead to other attacks (e.g., XSS, SQL injection if not properly handled).

## Target Endpoint

The target endpoints are typically where usernames are created or modified. In DVWA, this would primarily be during the initial setup/reset of the database (which creates default users) or if there were a user registration/profile update feature.

- `http://localhost/dvwa/setup.php` (Indirectly, as it creates default users)
- Hypothetical user registration or profile update pages.

## Methodology

The methodology involves attempting to create or modify usernames using various inputs that violate common security best practices. This includes trying:

- Very short or very long usernames.
- Usernames containing sensitive information (e.g., email addresses, social security numbers).
- Usernames with special characters, spaces, or non-standard encoding.
- Common or default usernames (e.g., "admin", "test", "user").

The script would observe if the application rejects these inputs or if it accepts them, potentially leading to security risks.

**Important Python Snippet:**

```
# Python snippet for this test would be similar to OTG-IDENT-002 (registration)
# or OTG-IDENT-003 (provisioning), focusing on the username input.
# Since DVWA doesn't have a direct user creation form for testing username policies,
# this test is often conceptual or relies on observing existing default users.
```

```
# Example (conceptual, as DVWA doesn't have a direct username
creation form):
# def create_user_with_username(session, username, password):
#     # Hypothetical endpoint for creating a user
#     create_user_url =
"http://localhost/dvwa/admin/create_user.php"
#     payload = {
#         "new_username": username,
#         "new_password": password,
#         "create": "Create User"
#     }
#     response = session.post(create_user_url, data=payload)
#     return response.status_code, response.text

# if __name__ == "__main__":
#     with requests.Session() as admin_session:
#         login(admin_session, "admin", "password") # Assume
admin is logged in
#
#         # Test with a very short username
#         status, content =
create_user_with_username(admin_session, "a", "password123")
#         print(f"Creating user 'a': Status {status}, Content:
{content}")
#         # Expected: Error about username length

#         # Test with special characters
#         status, content =
create_user_with_username(admin_session, "<script>alert(1)
</script>", "password123")
#         print(f"Creating user with XSS: Status {status},
Content: {content}")
#         # Expected: Input validation error
```

**Specific Library or Tools:**

- `requests`: For sending requests to user creation/modification endpoints.
- Burp Suite/OWASP ZAP: For manual testing and fuzzing username input fields.

## Step to Reproduce

1. **Identify Username Creation/Modification Points:** In DVWA, the default users are created during the database setup. If there were a user management interface, that would be the target.
2. **Test Username Length:**
    - Attempt to create a username that is extremely short (e.g., 1 character) or extremely long (e.g., 256+ characters).
    - **Observe Response:** Check if the application enforces minimum/maximum length requirements.
3. **Test Special Characters and Encoding:**
    - Attempt to create usernames with special characters (e.g., `!@#$%^&*()`), spaces, or non-standard Unicode characters.
    - Attempt to inject XSS payloads (e.g., `<script>alert('XSS')</script>`) or SQL injection payloads (e.g., `' OR 1=1--`) into the username field.
    - **Observe Response:** Check if the application properly validates and sanitizes these inputs.
4. **Test for Sensitive Information in Usernames:**
    - Attempt to create usernames that resemble sensitive data (e.g., email addresses, phone numbers, social security numbers).
    - **Observe Response:** While not always a direct vulnerability, allowing such usernames can increase the risk of information leakage or social engineering.
5. **Test for Common/Default Usernames:**
    - Observe if the application allows the creation of common usernames like "admin", "test", "user", "root".
    - **Observation:** DVWA by default uses "admin" and "user", indicating a weak policy in this regard.
6. **Log Evidence:** Record the usernames attempted, the application's response, and any error messages or unexpected behaviors.

## Log Evidence

```
# Example Log Evidence for OTG-IDENT-005

--- Test Case: OTG-IDENT-005 - Testing for Weak or unenforced
username policy ---
Target Endpoint: DVWA User Creation/Setup (Conceptual)
```

```
--- Observation: Default Usernames ---
DVWA uses default usernames: 'admin' and 'user'.
Observation: This indicates a weak username policy as these are
highly guessable.

--- Conceptual Test: Attempt to create username 'a' (too short)
---
(Assuming a user creation form exists)
Request: POST /create_user.php
Payload: new_username=a&new_password=...
Response: Error message: "Username must be at least 3
characters long."
Observation: Application enforces minimum username length.

--- Conceptual Test: Attempt to create username
'<script>alert(1)</script>' (XSS payload) ---
(Assuming a user creation form exists)
Request: POST /create_user.php
Payload: new_username=<script>alert(1)
</script>&new_password=...
Response: Error message: "Invalid characters in username." or
input sanitized.
Observation: Application performs input validation for special
characters.

--- Conclusion ---
DVWA's default usernames are weak. If a user creation mechanism
were present, further testing would be needed to confirm robust
username policy enforcement, including length, character sets,
and prevention of sensitive information.
```