Detailed Session Management Testing Report

Target URL: http://localhost:8080

Date: 2025-08-04

OTG-SESS-001: Testing for Bypassing Session Management Schema

Test Objective

This test checks if the application's session management can be bypassed through session ID prediction, manipulation, or accepting arbitrary session IDs. It verifies if session tokens are sufficiently random and unpredictable.

Key Python Snippet

```
def analyze_cookie(cookie_value):
    """Analyze cookie for common vulnerabilities."""
    analysis = {"length": len(cookie_value)}

    # Check if cookie is predictable
    if cookie_value.isdigit():
        analysis["predictable"] = True
    elif len(set(cookie_value)) < 5:  # Low entropy
        analysis["predictable"] = True
    else:
        analysis["predictable"] = False

    # Check encoding
    if "%" in cookie_value:
        analysis["encoded"] = True
```

```
    else:
        analysis["encoded"] = False

    return analysis
```

Target Endpoints

- Login: `/login.php`
- Profile: `/profile.php`

Methodology

1. Obtain a valid session ID through normal login
2. Analyze the session ID for predictability (length, character set, encoding)
3. Attempt to modify parts of the session ID and check if it's accepted
4. Generate random session IDs of similar length and check if they're accepted
5. Check if session ID can be passed via URL parameter

Steps to Reproduce

1. Login to application and capture session cookie
2. Modify last 4 characters of session ID and try accessing profile page
3. Generate random session ID with same length and try accessing profile page
4. Append session ID as URL parameter (`?session_id=...`) and try accessing profile page

Test Result

Status: No vulnerabilities found

Log Evidence


```
{
  "error": "Login failed"
}
```

OTG-SESS-002: Testing for Cookies Attributes

Test Objective

This test examines the security attributes of cookies (especially session cookies) including Secure, HttpOnly, SameSite flags, domain scope, and expiration policies. Proper cookie attributes are essential for preventing session hijacking.

Key Python Snippet

```
def get_cookie_attributes(response, cookie_name):
    """Get security attributes of a cookie."""
    cookie = response.cookies.get(cookie_name)
    if not cookie:
        return None

    return {
        "secure": cookie.secure,
        "httponly": "HttpOnly" in str(cookie),
        "samesite": "None",
        "domain": cookie.domain,
        "path": cookie.path,
        "expires": cookie.expires
    }
```

Target Endpoints

- Login: `/login.php`

Methodology

1. Perform login request and capture all cookies
2. Inspect each cookie for security attributes
3. Check for presence of Secure, HttpOnly, and SameSite flags
4. Verify domain and path restrictions are properly set
5. Check if session cookies have expiration dates (shouldn't)

Steps to Reproduce

1. Login to application while monitoring cookies in response
2. Inspect `PHPSESSID` and `security` cookies
3. Verify cookie attributes using browser developer tools or proxy

Test Result

Vulnerabilities Found:

- Cookie PHPSESSID has weak SameSite policy: None
- Cookie PHPSESSID has overly broad domain: localhost.local
- Cookie security has weak SameSite policy: None
- Cookie security has overly broad domain: localhost.local

Log Evidence


```
{
  "cookies_analyzed": {
    "PHPSESSID": {
      "secure": false,
      "httponly": false,
      "samesite": "None",
      "domain": "localhost.local",
      "path": "/",
      "expires": null
    },
    "security": {
      "secure": false,
      "httponly": false,
      "samesite": "None",
      "domain": "localhost.local",
      "path": "/",
      "expires": null
    }
  }
}
```


Recommendation

- Set `Secure` flag to ensure cookies are only sent over HTTPS
- Set `HttpOnly` flag to prevent JavaScript access
- Set `SameSite` to `Lax` or `Strict` to prevent CSRF
- Restrict cookie domain to specific subdomains if possible

OTG-SESS-003: Testing for Session Fixation

Test Objective

This test checks if the application is vulnerable to session fixation attacks, where an attacker can force a user to use a predetermined session ID. The application should regenerate session IDs after login.

Key Python Snippet

```
# Step 1: Get session before login
response = http_request("GET", LOGIN_URL)
session_cookie = next((c for c in response.cookies if
"session" in c.name.lower()), None)

# Step 2: Login with same session
cookies = {session_cookie.name: session_cookie.value}
login_response = http_request("POST", LOGIN_URL,
data=credentials, cookies=cookies)

# Step 3: Check if session ID changed
session_id_after =
login_response.cookies.get(session_cookie.name)
```

Target Endpoints

- Login: `/login.php`
- Profile: `/profile.php`

Methodology

1. Obtain a session ID before authentication
2. Use this same session ID during authentication
3. Check if the session ID remains the same after successful login
4. Verify if the pre-authentication session becomes authenticated

Steps to Reproduce

1. Visit login page and capture session cookie
2. Submit login credentials while preserving the same session ID
3. Check if session ID changes after successful login
4. Attempt to access protected resources with original session ID

Test Result

Status: No vulnerabilities found

> Note: The test couldn't find a session cookie before login, which prevented complete testing.

Log Evidence


```
{
  "error": "No session cookie found"
}
```


OTG-SESS-004: Testing for Exposed Session Variables

Test Objective

This test checks if session-related variables are exposed in client-side code, URLs, or error messages, which could lead to session hijacking.

Key Python Snippet

```
sensitive_patterns = [
    r'session["\']?[_:]?(id|token)\s*=\s*["\']?([^\s"\']+)',
    r'user["\']?[_:]?(id|token)\s*=\s*["\']?([^\s"\']+)',
    r'auth["\']?[_:]?(token|key)\s*=\s*["\']?([^\s"\']+)',
    r'                    ]*name=["\']?(session_?id|token)
["\']? [^>]*value=["\']?([^\s"\']+)',
    r'data-?session=["\']?([^\s"\']+)',
    r'window\.session(?:Id|Token)\s*=\s*["\']?([^\s"\']+)'
]
```

Target Endpoints

- Profile: `/profile.php`
- Dashboard: `/dashboard.php`
- Account: `/account.php`
- Settings: `/settings.php`

Methodology

1. Access authenticated pages after login
2. Search HTML source for session-related patterns
3. Check URL parameters for session tokens
4. Inspect JavaScript files and data attributes

Steps to Reproduce

1. Login to application
2. Visit various authenticated pages
3. View page source and search for session IDs/tokens
4. Check browser developer tools for exposed session data

Test Result

Status: No vulnerabilities found

Log Evidence


```
{
  "error": "Login failed"
}
```


OTG-SESS-005: Testing for Cross Site Request Forgery (CSRF)

Test Objective

This test checks if the application is vulnerable to CSRF attacks by examining forms for anti-CSRF tokens and verifying if actions can be performed without them.

Key Python Snippet

```
# Check if form has CSRF protection
has_csrf = any('csrf' in key.lower() or 'token' in key.lower() for key in form_data.keys())

# Test without CSRF token if form should have one
if has_csrf:
    modified_data = {k: v for k, v in form_data.items() if 'csrf' not in k.lower() and 'token' not in k.lower()}
    response = http_request(form_method, form_action, data=modified_data, cookies=cookies)
    if response and "success" in response.text.lower():
        results["vulnerabilities"].append(f"CSRF vulnerability in form at {form_action}")
```

Target Endpoints

- Sensitive Action: `/change_email.php`
- Other forms throughout the application

Methodology

1. Identify all forms in the application
2. Check for presence of CSRF tokens
3. Submit forms without CSRF tokens
4. Verify if actions succeed without proper tokens

Steps to Reproduce

1. Login to application
2. Access sensitive action page (e.g., change email)
3. Remove CSRF token from form submission
4. Submit form and check if action succeeds

Test Result

Status: No vulnerabilities found

Log Evidence

```
{
  "error": "Login failed"
}
```

OTG-SESS-006: Testing for Logout Functionality

Test Objective

This test verifies if the application properly invalidates sessions during logout by checking if session cookies are cleared and if back-button navigation is properly handled.

Key Python Snippet

```
# Step 1: Perform logout
logout_response = http_request("GET", LOGOUT_URL, cookies=cookies)

# Step 2: Try to access protected page after logout
profile_response = http_request("GET", PROFILE_URL, cookies=cookies)

# Step 3: Test back button after logout
back_response = http_request("GET", PROFILE_URL, cookies=cookies)

# Step 4: Test session cookie removal
if logout_response.cookies.get(session_cookie_name):
    logout_cookie = logout_response.cookies.get(session_cookie_name)
    if logout_cookie and logout_cookie.expires and logout_cookie.expires < time.time():
        results["session_invalidation_tests"].append("Passed: Session cookie expired on logout")
```

Target Endpoints

- Logout: `/logout.php`
- Profile: `/profile.php`

Methodology

1. Login to application and capture session cookie
2. Perform logout action
3. Attempt to access protected page with old session cookie

4. Check if session cookie is cleared or expired
5. Test back-button behavior after logout

Steps to Reproduce

1. Login to application
2. Click logout button
3. Attempt to navigate back or access protected pages
4. Check if session cookie still exists and is valid

Test Result

Status: No vulnerabilities found

Log Evidence


```
{
  "error": "Login failed"
}
```


OTG-SESS-007: Test Session Timeout

Test Objective

This test verifies if the application properly enforces session timeouts after a period of inactivity, preventing indefinite session validity.

Key Python Snippet

```
# Wait for session to expire (plus buffer time)
wait_time = SESSION_TIMEOUT + 60  # Session timeout + 1 minute
time.sleep(wait_time)
```

```
# Try to access protected page
response = http_request("GET", PROFILE_URL, cookies=cookies)

if response and "Welcome" in response.text:
    results["vulnerabilities"].append("Session timeout not enforced")
```

Target Endpoints

- Profile: `/profile.php`

Methodology

1. Login to application and capture session cookie
2. Wait for session timeout period plus buffer time
3. Attempt to access protected page with old session cookie
4. Verify if session is still valid

Steps to Reproduce

1. Login to application
2. Wait for more than the session timeout period (15 minutes + buffer)
3. Attempt to access protected page without new activity
4. Check if session is still valid

Test Result

Status: No vulnerabilities found

> Note: The test couldn't verify timeout enforcement due to login failure.

Log Evidence

```
{
  "error": "Login failed"
}
```

OTG-SESS-008: Testing for Session Puzzling

Test Objective

This test checks for session puzzling vulnerabilities where session variables set in one context are interpreted differently in another context, potentially leading to security issues.

Key Python Snippet

```
# Access a page that might store session variables
response = http_request("GET", f"{TARGET_URL}/cart.php?item=123", cookies=cookies)

# Access a different page that might interpret session variables differently
response = http_request("GET", f"{TARGET_URL}/checkout.php", cookies=cookies)

# Check if cart item is present (shouldn't be, if proper session isolation)
if "item=123" in response.text:
    results["vulnerabilities"].append("Session puzzling vulnerability detected")
```

Target Endpoints

- Cart: `/cart.php`
- Checkout: `/checkout.php`
- Admin: `/admin`

- Profile: `/profile.php`

Methodology

1. Set session variables in one context (e.g., cart page)
2. Access different context (e.g., checkout page)
3. Check if session variables carry over inappropriately
4. Test across privilege levels if possible

Steps to Reproduce

1. Login to application
2. Add item to cart (`/cart.php?item=123`)
3. Navigate to checkout page
4. Check if cart item appears in unexpected context
5. If admin access available, test session variables across privilege levels

Test Result

Status: No vulnerabilities found

Log Evidence


```
{
  "error": "Login failed"
}
```