Authentication Testing - Detailed Report

OTG-AUTHN-001: Testing for Credentials Transported over an Encrypted Channel

Test Objective

This test verifies whether authentication credentials and sensitive data are transmitted over secure, encrypted channels (HTTPS). It checks for mixed content issues and identifies endpoints that transmit sensitive data over unencrypted HTTP.

Key Python Snippet

```
def is_https(url):
    """Check if URL uses HTTPS."""
    return urlparse(url).scheme == "https"
```

Target Endpoints

- Login page: /login.php
- Logout: /logout.php
- Password reset: /reset_password.php
- Change password: /change_password.php
- Security questions: /security_questions.php
- Profile: /profile.php

Methodology

1. Check if login page uses HTTPS
2. Verify encryption status of all authentication-related endpoints
3. Scan login page for mixed content (HTTP resources loaded on HTTPS page)

Steps to Reproduce

1. Access the login page and check the URL protocol
2. Intercept network traffic using Burp Suite or browser developer tools
3. Verify if credentials are sent in plaintext or encrypted
4. Check all authentication-related endpoints for HTTPS usage

Log Evidence


```
{
  "login_encrypted": false,
  "other_auth_endpoints": {
    "logout": "HTTP",
    "password_reset": "HTTP",
    "change_password": "HTTP",
    "security_questions": "HTTP",
    "profile": "HTTP"
  },
  "vulnerabilities": [
    "Login credentials transmitted over unencrypted HTTP",
    "logout endpoint uses HTTP",
    "password_reset endpoint uses HTTP",
    "change_password endpoint uses HTTP",
    "security_questions endpoint uses HTTP",
    "profile endpoint uses HTTP",
    "Mixed content detected in login page"
  ]
}
```


OTG-AUTHN-002: Testing for Default Credentials

Test Objective

This test attempts to authenticate using common default username/password combinations to identify systems with unchanged default credentials.

Key Python Snippet

```
DEFAULT_CREDENTIALS = [
    ("admin", "admin"),
    ("admin", "password"),
    ("admin", "admin123"),
    ("root", "root"),
    ("test", "test"),
    ("user", "user"),
    ("administrator", "administrator"),
    ("guest", "guest"),
    ("demo", "demo"),
]
```

Target Endpoints

- Login page: /login.php
- Common admin paths: /admin, /administrator, /manager, /wp-admin

Methodology

1. Attempt login with each default credential pair
2. Check for common admin paths
3. Verify response for successful authentication indicators

Steps to Reproduce

1. Navigate to the login page
2. For each default credential pair, enter username and password
3. Check if login is successful (redirect to authenticated page, session cookie set)
4. Manually check common admin paths

Log Evidence


```
{
  "tested_credentials": [
    {
      "username": "admin",
      "password": "admin",
      "success": false
    },
```

```
    {
      "username": "admin",
      "password": "password",
      "success": false
    },
    ...
  ],
  "vulnerable_accounts": [],
  "vulnerabilities": []
}
```

OTG-AUTHN-003: Testing for Weak Lockout Mechanism

Test Objective

This test evaluates the account lockout mechanism by attempting multiple failed logins to determine if the system prevents brute force attacks.

Key Python Snippet

```
# Test with invalid credentials
for i in range(1, 11):  # Test up to 10 attempts
    data = {"username": TEST_USER[0], "password":
f"wrongpassword{i}"}
    response = http_request("POST", LOGIN_URL, data=data)

    if response and "invalid" in response.text.lower():
        results["failed_attempts"] = i
    else:
        break
```

Target Endpoints

- Login page: /login.php

Methodology

1. Attempt multiple failed logins (10 attempts in script)
2. Verify if account gets locked after threshold
3. Check if error messages reveal lockout status

Steps to Reproduce

1. Select a valid username
2. Attempt login with incorrect password multiple times
3. Observe system response after each attempt
4. Attempt valid login after multiple failures to check if account is locked

Log Evidence


```
{
  "failed_attempts": 0,
  "lockout_threshold": null,
  "vulnerabilities": [
    "Lockout mechanism not functioning properly"
  ]
}
```


OTG-AUTHN-004: Testing for Bypassing Authentication Schema

Test Objective

This test attempts to bypass authentication mechanisms through various techniques including direct page access, parameter tampering, and HTTP verb manipulation.

Key Python Snippet

```
# Parameter tampering
cookies = {"session": "authenticated=true; user=admin"}
response = http_request("GET", PROFILE_URL, cookies=cookies)

# HTTP verb tampering
methods = ["PUT", "DELETE", "PATCH"]
for method in methods:
    response = http_request(method, PROFILE_URL)
```

Target Endpoints

- Protected pages: /profile.php, /change_password.php, /admin/dashboard.php

Methodology

1. Attempt direct access to protected pages without authentication
2. Tamper with session parameters
3. Use alternative HTTP methods (PUT, DELETE, PATCH)
4. Test path traversal techniques

Steps to Reproduce

1. While logged out, attempt to access protected pages directly
2. Modify session cookies to include authentication flags
3. Use tools like Burp Suite to send requests with different HTTP methods
4. Test path traversal patterns in URLs

Log Evidence


```
{
  "methods_tested": [],
  "vulnerabilities": []
}
```

OTG-AUTHN-005: Test Remember Password Functionality

Test Objective

This test evaluates the "remember me" functionality by analyzing cookie attributes and persistence to identify insecure implementations that could lead to session hijacking.

Key Python Snippet

```
# Analyze cookies
for cookie in response.cookies:
    name = cookie.name
    value = cookie.value
    results["cookie_analysis"][name] = {
        "value": value,
        "secure": cookie.secure,
        "httponly": 'httponly' in (cookie._rest or {}),
        "samesite": (cookie._rest or {}).get("samesite")
    }
```

Target Endpoints

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

Methodology

1. Login with "remember me" enabled
2. Analyze cookie attributes (Secure, HttpOnly, SameSite)
3. Check for sensitive data in cookies
4. Test session restoration after browser restart

Steps to Reproduce

1. Login with "remember me" option checked
2. Inspect cookies using browser developer tools
3. Verify cookie security flags
4. Close and reopen browser to check if session persists

Log Evidence


```
{
  "cookie_analysis": {
    "PHPSESSID": {
      "value": "7ieivpa51p4snvn707bjqch194",
      "secure": false,
      "httponly": false,
      "samesite": null
    },
    "security": {
      "value": "low",
      "secure": false,
      "httponly": false,
      "samesite": null
    }
  },
  "vulnerabilities": []
}
```


OTG-AUTHN-006: Testing for Browser Cache Weakness

Test Objective

This test checks if sensitive authentication data is cached by the browser, which could allow unauthorized access through browser history or cache.

Key Python Snippet

```
# Check cache headers
headers = response.headers
```

```
cache_related = ["Cache-Control", "Pragma", "Expires"]

for header in cache_related:
    if header in headers:
        results["cache_headers"][header] = headers[header]
```

Target Endpoints

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

Methodology

1. Access authenticated pages
2. Check Cache-Control, Pragma, and Expires headers
3. Test back button behavior after logout
4. Verify if sensitive data remains in browser cache

Steps to Reproduce

1. Login and access sensitive pages
2. Inspect network traffic for cache headers
3. Logout and use browser back button
4. Check if sensitive pages are accessible from cache

Log Evidence


```
{
  "error": "Failed to access profile"
}
```


OTG-AUTHN-007: Testing for Weak Password Policy

Test Objective

This test evaluates the strength of password policies by attempting to register and change passwords using weak, common passwords.

Key Python Snippet

```
WEAK_PASSWORDS = [
    "password", "123456", "qwerty", "letmein", "welcome",
    "admin123", "passw0rd", "12345678", "abc123",
"Password1"
]
```

Target Endpoints

- Registration page: /register.php
- Change password: /change_password.php

Methodology

1. Attempt registration with weak passwords
2. Try changing password to weak alternatives
3. Verify if system enforces complexity requirements

Steps to Reproduce

1. Navigate to registration page
2. Attempt to register with weak passwords
3. After successful registration, attempt to change to weak password
4. Verify if system accepts weak passwords

Log Evidence


```
{
  "error": "Registration page not found"
}
```

OTG-AUTHN-008: Testing for Weak Security Question/Answer

Test Objective

This test evaluates the strength of security questions by attempting to guess answers using common responses and brute force techniques.

Key Python Snippet

```
COMMON_SECURITY_QUESTIONS = {
    "What was your first pet's name?": ["Fluffy", "Max",
"Bella"],
    "What is your mother's maiden name?": ["Smith",
"Johnson", "Williams"]
}
```

Target Endpoints

- Password reset: /reset_password.php
- Security questions: /security_questions.php

Methodology

1. Initiate password reset process
2. Identify security question
3. Attempt common answers
4. Brute force with random answers

Steps to Reproduce

1. Start password reset for a known account
2. Note the security question presented
3. Try common answers associated with the question
4. Verify if system accepts weak answers

Log Evidence

```
{
  "status": "Security questions not implemented"
}
```

OTG-AUTHN-009: Testing for Weak Password Change or Reset Functionalities

Test Objective

This test evaluates the security of password reset mechanisms by checking for token exposure, predictability, and account takeover vulnerabilities.

Key Python Snippet

```
# Test token predictability
predictable_token = "000000"
reset_url = f"{RESET_PASSWORD_URL}?token=
{predictable_token}"
response = http_request("GET", reset_url)
```

Target Endpoints

- Password reset: /reset_password.php
- Profile: /profile.php

Methodology

1. Check for token exposure in response
2. Test predictable token patterns
3. Attempt account takeover via email change
4. Verify token expiration and reuse

Steps to Reproduce

1. Initiate password reset
2. Inspect response for token exposure
3. Attempt to guess reset token
4. Change email address and attempt password reset

Log Evidence

```
{
  "error": "Password reset request failed"
}
```

OTG-AUTHN-010: Testing for Weaker Authentication in Alternative Channel

Test Objective

This test checks if alternative authentication channels (mobile API, web services) have weaker security controls than the main web interface.

Key Python Snippet

```
alternative_channels = [
    {"name": "Mobile API", "url": f"
{TARGET_URL}/api/v1/login", "method": "POST"},
    {"name": "Web Services", "url": f"
{TARGET_URL}/webservice/login", "method": "POST"}
]
```

Target Endpoints

- Mobile API: /api/v1/login

- Web services: /webservice/login
- Mobile site: /m/login

Methodology

1. Identify alternative authentication channels
2. Test with default credentials
3. Check for missing MFA
4. Verify session management across channels

Steps to Reproduce

1. Discover alternative authentication endpoints
2. Attempt authentication with weak credentials
3. Check if MFA is enforced
4. Test if web session works on alternative channels

Log Evidence


```
{
  "channels_tested": [],
  "vulnerabilities": []
}
```