



Entity Authentication and Session Management

Jim Manico

@manicode

OWASP Volunteer

- *Global OWASP Board Member*
- *OWASP Cheat-Sheet Series, Top Ten Proactive Controls, OWASP Java Encoder and HTML Sanitizer Project Manager and Contributor*

Secure-Coding Instructor/Author

- *16 years of web-based, database-driven software development and analysis experience*
- *Working on a Java Security book with McGraw-Hill and Oracle Press!*

Kama'aina Resident of Kauai, Hawaii



Where are we going?

Authentication

Session Management

Transport Security

Password Storage

Multi-Factor Authentication

Forgot Password Workflow

What is Entity Authentication?

What is Authentication

- Verification that an entity is who it claims to be.

Difference between Authentication and Authorization

- Authorization is checking if an entity has privileges to perform a function/action whilst Authentication is verification of identification.

What is a Authentication Session?

A session identifier (ID) is supplied to the entity once they are authenticated.

This is a random, unique & difficult to guess string.

■ ASEIUHF849J283JE874GSJWOD2374DDEOFEFK93423H

It is used by the entity on any subsequent communication to identify the source of the messages

It is valid for a finite period of time

We need a session ID as HTTP is stateless, it has no memory

The session ID is a “key” to a portion of memory on the server where your individual data and/or state can be stored

Entity Authentication Workflow

- 1 Start HTTPS, deliver Login form
- 2 Submit Credentials
- 3 Create Session, Deliver session cookie to user
- 4 Do cool things
- 5 Potential Re-Authentication
- 6 Absolute Timeout
- 7 Logoff or Idle Timeout
- 8 Invalidate Session back to HTTP if desired

Session Identifiers

Once a user has proven their identity, session management functionality is employed

Each request sent to the server contains an identifier that the server uses to associate requests to a specific authenticated user

The session ID is often all that is needed to prove authentication for the rest of the session

A stolen active session ID allows an attacker to hijack a logged-in account (but does not reveal the victims credentials)

Session ID's are typically passed in a HTTP Cookie

In general, this is transparent to the developer and is handled by web frameworks

Authentication Dangers

Passwords & PIN's

- Database stolen revealing stored password data
- Brute force attack attempting many password attempts for a specific account
- Simple password policy allowing faster guesses
- Password reuse: attacks on one website effect others

Username Harvesting

- Registration page often makes this easy

Weak "Forgot Password" feature

- Reset links sent over email

More Authentication Dangers

Weak "Change Password" feature

- Does not require existing password
- Access control weakness allows reset of other users password

Session Management Dangers

- Forcing victims to use known session ID's
- Weak or predictable session ID's
- Session Hijacking via XSS
- Session Hijacking via network sniffing
- Lack of session timeout, sessions that never expire

Credential Security

Should require the user to provide proof of identity (re-authentication)

- Login
- Change Password
- Changing email address
- Significant or anomalous transactions
- Helps minimize CSRF and session hijacking attacks

Implement server-side enforcement of password syntax and strength (i.e. length, character requirements, etc)

- Tough balance, overly strong policy is bad
- Do not allow users to choose commonly used passwords!

Re-authentication Examples

Change E-mail

Use the form below to change the e-mail address for your Amazon.com account. Use address next time you log in or place an order.

What is your new e-mail address?

Old e-mail address: jim@manico.net

New e-mail address:

Re-enter your new e-mail address:

Password:

[Save changes](#)

Change Your Email Address

Current email: jim@manico.net

New email

Meetup password

[Submit](#)

[Cancel](#)

[Forgot your password?](#)

Primary email: ☒ jim@manico.net

New Email:

Facebook email: jmanico@facebook.com

Your Facebook email is based on your public username. Email sent to this address goes to Facebook Messages.

☐ Allow friends to include my email address in [Download Your Information](#)

To save these settings, please enter your Facebook password.

Password: ✗ Wrong password.

[Save Changes](#) [Cancel](#)

Save account changes ✕

Re-enter your Twitter password to save changes to your account.

[Forgot your password?](#)

[Cancel](#) [Save changes](#)

Information to reset my password

a password reset by entering only your

this box, you will be prompted to enter

ne number if you forget your password.

Setting is saved to this browser.

You can request a file containing your information, starting with your first Tweet. A link will be emailed to you when the file is ready to be downloaded.

Login and Session Security

Send all credentials and session id's over well configured HTTPS/SSL/TLS

- Helps avoid session hijacking via network sniffing

Develop generic failed login messages that do not indicate whether the user-id or password was incorrect

- Minimize username harvesting attack

Enforce account lockout after a pre-determined number of failed login attempts

- Stops brute force threat
- Account lockout should trigger a notification sent to application administrators and should require manual reset (via helpdesk)

Cookie Options

The Set-Cookie header uses the following syntax:

**Set-Cookie: NAME=VALUE; expires=DATE; path=PATH;
domain=DOMAIN_NAME; secure; httponly;**

Name

- The name of the cookie parameter

Value

- The parameter value

Expires

- The date on which to discard the cookie



Cookie Security Defenses

Path	The path under which all requests should receive the cookie. "/" would indicate all paths on the server
Domain	The domain for which servers should receive the cookie (tail match). For example, my.com would match all hosts within that domain (www.my.com, test.my.com, demo.my.com, etc.)
Secure	Indicates that the cookie should only be sent over HTTPS connections
HTTPOnly	Helps ensure Javascript can not manipulate the cookie. Good defense against XSS

Additional Cookie Security Defenses

Avoid storing sensitive data in cookies

Avoid using persistent cookies

Any sensitive cookie data should be encrypted if not intended to be viewed/tampered by the user. Persistent cookie data not intended to be viewed by others should always be encrypted.

Cookie values susceptible to tampering should be protected with an HMAC appended to the cookie, or a server-side hash of the cookie contents (session variable)

Additional Session Defences

Generate new session ID at login time

- To avoid *session fixation* threat

Session Timeout (sessions must “expire”)

- Idle Timeout due to inactivity
- Absolute Timeout
- Logout Functionality
- Will help minimize session hijacking threat

Session Management Code Review Challenge

Challenge!

Examine the following pseudo code and identify any issues with this session management mechanism

Pseudo Code: Session Creation, Authentication, Session Validation

ROW	CODE	FIX? Y/N
1	BROWSER requests access to "Account Summary" from WEBSERVER	
2	WEBSERVER checks whether the session is authenticated	
3	IF session is authenticated:	
4	Send "Account Summary" page to BROWSER	
5	RETURN	
6	IF session is NOT authenticated:	
7	WEBSERVER grabs USERNAME posted by BROWSER	
8	WEBSERVER asks DATABASE ("Select * from AuthTable where Username = '%s'", USERNAME);	
9	IF DATABASE returns no users:	
10	WEBSERVER sends error message to BROWSER ("Invalid User Name %s", USERNAME);	
11	RETURN	
12	ELSE	
13	WEBSERVER grabs PASSWORD posted by BROWSER	
14	For each user returned by DATABASE:	
15	IF user's password equals PASSWORD:	
16	Authenticate session	
17	Generate Session ID:	
18	Increment previous Session ID by 1	
19	Store Session ID	
20	Add Session ID to user's cookie	
21	IF no users have a password equal to PASSWORD:	
22	WEBSERVER sends error message to Browser ("Invalid password %s for username %s", PASSWORD, USERNAME);	

Solution

1	BROWSER requests access to "Account Summary" from WEBSERVER	
2	WEBSERVER checks whether the session is authenticated	
3	IF session is authenticated and the user has access to "Account Summary"	
4	Send "Account Summary" page to BROWSER	
5	RETURN	
6	IF session is NOT authenticated:	
7	WEBSERVER grabs USERNAME and PASSWORD posted by BROWSER	
8	WEBSERVER CREATES BCrypt OF PASSWORD EVEN IF USERNAME DOES NOT EXIST	
9	WEBSERVER asks DATABASE ("select * from AuthTable where Username = '%s' and PasswordBcrypt = '%s'", USERNAME, BCrypt OF PASSWORD);	
10	IF DATABASE returns no users or more than one user:	
11	WEBSERVER sends error message to BROWSER ("Invalid User Name or Password");	
12	RETURN	
13	ELSE (DATABASE has returned exactly one user)	
14	Authenticate session	
15	Generate Session ID:	
16	WEBSERVER generates secure random Session ID	
17	Store Session ID	
18	Add Session ID to user's SECURE, HTTPONLY cookie	

Logout/Session Defences

Give users the option to log out of the application and make the option available from every application page

When clicked, the logout option should prevent the user from requesting subsequent pages without re-authenticating to the application

The user's session should be terminated using a method such as `session. abandon()`, `session. invalidate()` during logout

JavaScript can be used to force logout during window close event

Password Defenses

- Disable Browser Autocomplete

- ▶ `<form AUTOCOMPLETE="off">`
- ▶ `<input AUTOCOMPLETE="off">`

- Only send passwords over HTTPS POST

- Do not display passwords in browser

- ▶ `Input type=password`

- Store password quickly verifiable but not reversible

- ▶ Use a Salt
- ▶ SCRYPT/PBKDF2
- ▶ HMAC

Password Storage in the Real World

1. Do not limit the characters or length of user password
2. Do not allow users to use commonly used passwords
3. Use a user-specific salt
4. Store passwords as an HMAC + good key management
5. Use SCRYPT or PBKDF2 as an alternative

1) Do not limit the password strength!

- Limiting passwords to protect against injection is doomed to failure
- Use proper encoding and other defenses instead
- Very long passwords can cause DOS
- **Limit commonly used passwords!**

Password1!

2) Use a user-specific salt!

- **protect**([salt] + [credential]);
- Use a 32+ byte salt
- Do not depend on hiding, splitting, or otherwise obscuring the salt
- Consider hiding, splitting or otherwise obscuring the salt anyway as a extra layer of defense

3) Leverage keyed protection solution

- **HMAC-SHA-256([key], [salt] + [credential])**
- Protect this key as any private key using best practices
- Store the key outside the credential store
- Isolate this process outside of your application layer

**Imposes difficult verification on
the attacker, only!**

3) Leverage an adaptive one-way function

- **PBKDF2**([salt] + [password], c=10,000,000);
- **PBKDF2** when FIPS certification or enterprise support on many platforms is required
- **B/Script** where resisting any/all hardware accelerated attacks is necessary but support isn't

**Imposes difficult verification on the
attacker *and* defender!**

Forgot Password Secure Design

Require identity questions

- Last name, account number, email, DOB
- Enforce lockout policy

Ask one or more good security questions

- https://www.owasp.org/index.php/Choosing_and_Using_Security_Questions_Cheat_Sheet

Send the user a randomly generated token via out-of-band communication

- email, SMS or token

Verify code in same web session

- Enforce lockout policy

Change password

- Enforce password policy

Federated Identity and SAML

XML-based identity management between different businesses

Centralized Authentication Authority

Single Sign On / Single Logout

Assertions and Subjects

Authentication Assertion Types

Attribute Assertion Types

Entitlement Assertion Types

Multi Factor Authentication

There are 3 methods of identifying an individual

- Something you have – e.g. token, certificate, cell
- Something you are – e.g. biometrics
- Something you know – e.g. password.

For highly sensitive applications multifactor authentication can be used

Financial services applications are moving towards “stronger authentication”

Google/Facebook/World-Of-Warcraft support consumer-centric multi-factor authentication

Multi Factor Authentication



**Google, Facebook, PayPal, Apple, AWS, Dropbox, Twitter
Blizzard's Battle.Net, Valve's Steam, Yahoo**

Basic MFA Considerations

■ Where do you send the token?

- ▶ Email (worst)
- ▶ SMS (ok)
- ▶ Mobile native app (good)
- ▶ Mobile native app, push notification (great)
- ▶ Dedicated token (ideal)
- ▶ Printed Tokens (interesting)

■ How do you handle thick clients?

- ▶ Email services, for example
- ▶ Dedicated and strong per-app passwords

Basic MFA Considerations

■ How do you handle unavailable MFA devices?

- ▶ Printed back-up codes
- ▶ Fallback mechanism (like email)
- ▶ Call in center

■ How do you handle mobile apps?

- ▶ When is MFA not useful in mobile app scenarios?

Authentication Control Flow Flaws



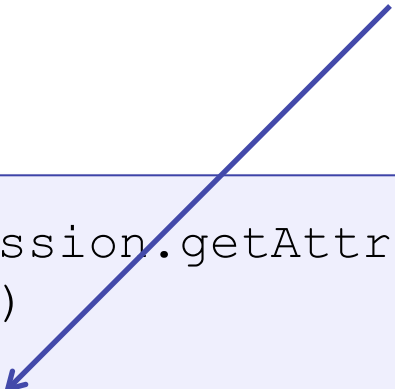
Does this code look *safe* to you?

```
String username = session.getAttribute("user");  
if (username == null)  
{  
    response.sendRedirect("Access Denied");  
}
```

Business Logic Processing

Authentication Control Flow Flaws

What if the execution did not stop **here**?



```
String username = session.getAttribute("user");  
if (username == null)  
{  
    response.sendRedirect("Access Denied");  
}
```

Business Logic Processing

Control Flow Flaws

- Business logic would execute for an *unauthenticated request*!

```
String username = session.getAttribute("user");  
if (username == null)  
{  
    response.sendRedirect("Access Denied");  
}
```

Business Logic Processing

This is not protected

Control Flow Flaws

- The **execution flow** does not **stop** after the *response.sendRedirect* call
- Entire page is processed and then the user is redirected to error page
- Thus, the business logic remains unprotected

Return after redirecting!

■ Security Measures:

- Terminate the execution flow after redirection call.

```
String username = session.getAttribute("user");  
if (username == null)  
{  
    response.sendRedirect("Access Denied");  
    return;  
}
```

Business Logic Processing

Other Authentication Considerations

- Do password hashing IN THE BROWSER to prevent password theft even when SSL is compromised
- List all logins with login time, logout time, IP address or more
- Provide login auditing in a READ ONLY way
- Provide capability to block logins from certain geographic regions
- Provide capability to block logins during certain times
- Do not allow a password reset workflow that could be compromised if a customers email was popped
- Disallow commonly used passwords, even ones that fit a strong password policy (like Password1!)

Summary

Authentication

Session Management

Transport Security

Password Storage

Multi-Factor Authentication

Forgot Password Workflow