Entity Authentication and Session Management

SecAppDev 2013

The OWASP Foundation
http://www.owasp.org

What is Entity Authentication?

What is Authentication

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

Difference between Authentication and Authorisation

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

Entity Authentication Basics

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

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 memnory on the server where your individual data and/or state can be stored

Entity Authentication Workflow

1. Login form
2. Submit Credentials
3. Create Session
4. Do cool things
5. Potential Re-Authentiation
6. Absolute Timeout
7. Logoff or Idle Timeout
8. Invalidate Session

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 authenticated users
- The session ID is all that is needed to prove authentication for the rest of the session
- A stolen active session ID is similar to having your credentials stolen
- Session ID's are typically passed in a HTTP Cookie
- In general, this is transparent to the developer and is handled by the web framework

Authentication Dangers

Passwords & PIN's	■ Plaintext or poor password storage ■ Subject to brute-force attack ■ Weak Password Policy ■ Password reuse
Username Harvesting	■ Registration page 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	■ Session Fixation ■ Weak or Predictable Session ■ Session Hijacking via XSS ■ Session Hijacking via network sniffing ■ Lack of idle and absolute session timeout

Credential Defenses

Sensitive or should require the user to provide proof of identity

- Login
- Password Reset
- Shipping to a new address
- 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

Login and Session Defenses

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)

More Session Defenses

Ensure that Session ID’s are protected in a HTTP Cookie

- Secure, HTTP Only, limited path

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

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 (if absent, the cookie not persistent and is discarded when the browser is closed

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.

Cookie Security Defenses

- Avoid storing sensitive data in cookies
- Avoid using persistent cookies
- Always set the "secure" cookie flag for HTTPS cookies to prevent transmission of cookie values over unsecured channels
- 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)

Logout/Session Defenses

- 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
  - Do not display passwords in HTML document

- Store password on server via one-way encryption
  - Hash password
  - Use Salt
  - Iterate Hash many times
  - BCRYPT/PBKDF2

Password Storage Code Sample

```
public String hash(String plaintext, String salt, int iterations)
     throws EncryptionException {
byte[] bytes = null;
try {
  MessageDigest digest = MessageDigest.getInstance(hashAlgorithm);
  digest.reset();
  digest.update(ESAPI.securityConfiguration().getMasterSalt());
  digest.update(salt.getBytes(encoding));
  digest.update(plaintext.getBytes(encoding));
  // rehash a number of times to help strengthen weak passwords
  bytes = digest.digest();
  for (int i = 0; i < iterations; i++) {
     digest.reset();  bytes = digest.digest(salts + bytes + hash(i));
   }
  String encoded = ESAPI.encoder().encodeForBase64(bytes,false);
  return encoded;
} catch (Exception ex) {
       throw new EncryptionException("Internal error", "Error");
}}
```

Forgot Password Secure Design

Require identity questions

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

Ask one or more good security questions

- http://www.goodsecurityquestions.com/

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

Encryption in Transit (TLS)

Authentication credentials and session identifiers must me be encrypted in transit via HTTPS/SSL

- Starting when the login form is rendered
- Until logout is complete
- All other sensitive data should be protected via HTTPS!

https://www.ssllabs.com free online assessment of public facing server HTTPS configuration

https://www.owasp.org/index.php/Transport Layer Protection Cheat Sheet for HTTPS best practices

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

Session Management Code Review Challenge

Challenge!

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

Pseudo Code: Session Creation, Authorization, 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:	
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 asks DATABASE ("Select * from AuthTable where Username = '%s' **and Password = '%s'**", USERNAME, **PASSWORD**);	
9	IF DATABASE returns no users **or more than one user:**	
10	WEBSERVER sends error message to BROWSER ("Invalid User Name **or Password");**	
11	RETURN	
12	ELSE (DATABASE has returned exactly one user)	
13	Authenticate session	
14	Generate Session ID:	
15	**WEBSERVER generates secure Session ID**	
16	Store Session ID	
17	Add Session ID to user's cookie	