A2: Broken Authentication and Session Management

But first...

Authentication, Authorization, Sessions
Authentication

- Determining user identity
- Multiple ways
  - What you know (password)
  - What you have (phone, RSA SecureID)
  - Who you are (fingerprint, eye scan)
  - Where you are (GPS, IP address)
Authorization

- Ensure users only perform actions within their privilege level or role (A4/A7)
- **Policy** to determine which users are allowed which actions on which objects
  - Users
    - Entity making request on resource
    - User, external web application, internal web application, database
  - Actions
    - Read, Write, Execute, Append, Create, Delete
  - Objects
    - Resource layers being protected
    - Network, operating system, files, web application, database, etc.
  - Policy
    - Kinds of access control
    - Discretionary Access Control (object owner decides)
    - Mandatory Access Control (system/administrator decides)
      - Stronger limits on activity
      - Role-Based Access Control (system decides based on user role)
Session management

- Embodiment of user’s authentication and authorization for duration of the user’s interaction with service
- Sessions used to maintain authentication and authorization state over stateless HTTP
- Done via multiple mechanisms sent on each request
  - HTTP cookies
  - URL parameters
  - Flash Local Shared Objects
  - JavaScript web tokens
  - HTML5 session storage
  - Hidden Form fields
A2 – Broken Authentication and Session Management
Example: Guessable credentials

- Common passwords and weak passwords allowed

<table>
<thead>
<tr>
<th>RANK</th>
<th>PASSWORD</th>
<th>CHANGE FROM 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123456</td>
<td>Unchanged</td>
</tr>
<tr>
<td>2</td>
<td>password</td>
<td>Unchanged</td>
</tr>
<tr>
<td>3</td>
<td>12345678</td>
<td>1 ↪</td>
</tr>
<tr>
<td>4</td>
<td>qwerty</td>
<td>1 ↪</td>
</tr>
<tr>
<td>5</td>
<td>12345</td>
<td>2 ↩</td>
</tr>
<tr>
<td>6</td>
<td>123456789</td>
<td>Unchanged</td>
</tr>
<tr>
<td>7</td>
<td>football</td>
<td>3 ↩</td>
</tr>
<tr>
<td>8</td>
<td>1234</td>
<td>1 ↩</td>
</tr>
<tr>
<td>9</td>
<td>1234567</td>
<td>2 ↩</td>
</tr>
<tr>
<td>10</td>
<td>baseball</td>
<td>2 ↩</td>
</tr>
<tr>
<td>11</td>
<td>welcome</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1234567890</td>
<td></td>
</tr>
</tbody>
</table>

- Allows brute-force attacks by an adversary
- Default passwords or security credentials left unchanged
  - http://www.defaultpassword.com/
  - Mirai, Dyn IoT attack (10/2016)
- Guessable password reset questions
  - Information publicly available or easily inferred
  - Anonymous hack of Sarah Palin’s Yahoo account 2008
    - ZIP, birthdate, where she met spouse
Example: Vulnerable authentication processes

- Predictable change my password links
- No rate-limits on authentication attempts and failures
  - Via web front-end and web API
- Side-channel attacks
  - Non-time-constant string comparison vulnerability (Program #2)
  - Username checked before password instead of simultaneously
Example: Password storage problems

- Passwords stored in the clear instead of hashed
  - Single security compromise gives up all user credentials
  - Credential reuse makes problem worse

- Password hashes used, but stored without a “salt”
  - Salt is random data hashed with password
  - Attacker can employ precomputed dictionary attack via rainbow tables
  - Rainbow table lookup https://crackstation.net
Example: Password storage problems

- Salt prevents rainbow table lookup

- But, cryptographic hashes used instead of password hashes
  - Cryptographic hashes intended to be *fast*
  - But, if one has salt and hash, a brute-force dictionary attack is *still* fast against weak passwords
Example: Password storage problems

- Ideal:
  - Hashes that are extremely slow to compute
  - Attacker obtaining hashes can’t perform an efficient, offline dictionary attack to obtain weak passwords of users
- One mechanism: use a salt and iterate through a password hash algorithm multiple times
  - scrypt or bcrypt (iteration = 100ms)
  - PBKDF2([salt] + [password], c=140,000);
  - https://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security
Example: Session IDs carried in URLs

1. User sends credentials

2. Site uses URL rewriting (i.e., put session in URL)

3. User clicks on a link to [http://www.hacker.com](http://www.hacker.com) in a forum

4. Hacker checks referrer logs on [www.hacker.com](http://www.hacker.com) and finds user’s JSESSIONID

5. Hacker uses JSESSIONID and takes over victim’s account
Example: Exposed tokens and cookies

- Cookies sent over HTTP
  - Dump via Burp, Wireshark, or browser tools
  - Session hijacking, request forgery
Example: Vulnerable tokens and cookies

- Repeated or unchanging session tokens
  - Persistent access if captured
- Predictable generation of session tokens
  - Blind hijacking of authorized sessions
- Unsigned session tokens
  - Forging authorized sessions (JavaScript Web Token)
Example: Vulnerable tokens and cookies

- Insecure management of session information at server
  - User sessions stored in server insecurely
  - PHP active sessions directory
    ```
    # cat /var/lib/php5/sess_o8d7lr4p16d9gec7ofkdbnhm93pentesterlab|s:12:"pentesterlab";
    ```
- Global session management in web frameworks
  - Vulnerable to side-channel attacks for co-located apps (natas)
Example: Case sensitivity mismatch

- Database and web application handle case differently
- Creating a user with an existing username
- Allow access to “admin” account via “Admin” or “ADMIN”
A2 – Prevention

Authentication Cheat Sheet
https://www.owasp.org/index.php/Authentication_Cheat_Sheet

Password Storage Cheat Sheet
https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

Forgot Password Cheat Sheet
https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet

Session Management Cheat Sheet
https://www.owasp.org/index.php/Session_Management_Cheat_Sheet
Verify authentication architecture

- Leverage session management provided by your framework
- Ensure SSL/TLS protects all credential transmission
  - Form data, HTTP Auth, etc.
- Limit exposure of credentials (e.g. do not send repeatedly)
- Ensure secure storage of credentials at end points
Password management best practices

- Remove unnecessary accounts and default credentials
- Enforce strong password policies
  - Ensure capital+lower-case letters, numbers, symbols used
  - No username in password
  - No dictionary words in password
  - Enforce minimum length > 8 characters
  - No obvious substitutions (e.g. zero for o)
  - No common passwords
- Ban credentials that have been compromised and dumped
  - https://haveibeenpwned.com/
- Recommend password managers to users
- Employ hash stretching and a password hashing algorithm when storing passwords (described previously)
Multi-factor authentication

- Employ out-of-band token
  - Two-factor auth via Google Authenticator, Duo, RSA SecureID
  - SMS or other mobile messaging app
- Biometric authentication
- Use IP address and geographic location information
- Multiple, good identity questions
- Enforce lockout policy on multiple failures
- Employ security seals in authentication
  - To train users to detect phishing attacks
Authorization best practices

- Centralize authorization mechanism
- Minimize custom authorization code
- Authorize every request at the server
- Fail closed
  - Unexpected input causes connection termination (see PHP issues)
- Operate with Least Privilege
  - Separate user and administrator accounts
  - Run web server as a regular user
- Keep accounts unique
  - No account sharing
Session management architecture

- Keep as much information on server as possible
  - Rely upon an opaque session ID
- Never trust client or network
  - Avoid insecure client-side authorization tokens
  - Encrypt and digitally sign anything sent to client for storage that needs to come back unmodified (while keeping key to yourself)
  - Remove session information from URL (and thus, browsing history)
- Timeout sessions
  - Ensure session ID expiration
  - Verify that logoff actually destroys the session (OWASP’s WebScarab)
- Ensure all session information transmitted via SSL/TLS and only via HTTPS
  - e.g. secure flag and HTTP-only flag on cookies
Labs, homework, and program

- See handout
- Session #2
  - For AJAX responses, in Chrome
    - Developer Tools:Network:XHR:<req>:Response
- Session #4
  - In python3
    
    \[
    \text{username} = \text{"%04d"} \ % \ i
    \]
Extra
Example: Poor CAPTCHAs

- **Logic flaws**
  
  ```ruby
  if params[:captcha] and params[:captcha] != session[:captcha]
    # ERROR: CAPTCHA is invalid redirect
  end
  # CAPTCHA is valid
  # If no captcha is provided, the script does not fail safely
  ```

- CAPTCHA answer easily guessable
  - Single-digit sum
  - Repeated answers
  - OCR tools
  - Improper rate-limits for incorrect guesses