#7 – Broken Authentication Exploitation

What makes an application vulnerable?

- Credentials can be **guessed** or **overwritten** because of weak account management functions
  - Default passwords, weak passwords (low entropy), predictable passwords
  - Broken account creation/recovery process

- Allow **brute-force** or **dictionary attacks** (*credential stuffing*)
  - Test the entire key space by blind variation of characters
  - Test the entire key space by testing all

What makes an application vulnerable?

- Credentials are stored in a vulnerable format
  - Clear text in a database, weak cipher mode, low PBKDF2 rounds
  - Attacker can conduct an **offline attack**
    - Offline attacks are dangerous:
      - The victim doesn't know the resources of the attacker
      - The attack is silent to the victim and can take days-years
- Credentials are sent over **unencrypted connections**
  - Or authentication driven tokens (cookies)
- Recovery or Multi-Factor is broken/missing

What applications should do

Avoid passwords as much as possible (CWE-309)

unpredictability

- Passwords are prone to have low entropy, follow patterns
  - If passwords are required, force some entropy (CWE-521)
- Users frequently **reuse credentials** among different services
- Passwords must be stored in the server
  - May also be stored in the client

What applications should do

Use secure storage (CWE-257)

- Do **not store passwords in clear**, even if the domain is “secure”
  - E.g. database requires authentication to be accessed
- Add a **computational/storage complexity** transformation function (CWE 916)
  - PKBDF2 or scrypt
  - Use a reasonably high number of blocks/Rounds
- Direct access to storage may reveal secrets
  - Directly of through key brute force

What applications should do

Require rotation, but don't require frequent rotation, except if compromises are recorded (CWE-263)

- Rotation is important and will impose expiration on secrets (CWE-362)
  - Stolen/discovered secrets will be rendered useless
  - Doesn't depend on users good practices (**it's imposed by system**)
- Frequent rotation without proper tools will be rendered useless as **users will create "algorithms"**
  - 01MocKingBird2020, 02MocKingBird2020...
  - Frequent expiration will impact usability and increase the security burden

What applications should do

Rate limit authentication functions (CWE-770)

- Password stuffing will be **dramatically delayed**
  - Even a small delay of hundreds of milliseconds may be useful
- **Monitoring authentication functions** allow detecting attack attempts (CWE-307)
  - Blocking an account after repeated authentication failures
  - Password Spraying may circumvent methods (CAPEC-565)

What applications should do

Use Multi Factor Authentication (CWE-308)

- The attack required to obtain a credential, may not obtain a smartphone, or a hardware token
  - Credential: eavesdropping or database
  - Smartphone: remote compromise or physical steal
  - Hardware token: physically steal the token
- If it is a usability issue, use progressive multi factors
  - E.g.: Check <secret, cookie and IP network> and a fourth is something changes
- Favor multi-factor authentication in recovery processes

What applications should do

- For reference:
- NIST 800-63B: Authentication Assurance Levels

Token exploitation

- Client may **freely manipulate tokens** to trigger an attack
  - Break the authentication process, Enumerate users, Bypass authentication
- Cookies
  - If contain sensitive information (passwords) – CWE-256
  - If they have low entropy
  - If they have a structure that is processed in the server
- JWT
  - If server **improperly verifies signature** and allows the “none” method
    - Verification method code must enforce the same method used in the signature creation
  - Short secret allows an attacker to **forge tokens**

Session Hijacking

- Web applications use session ids, cookies and tokens as credentials
  - Stealing this credential will result in **session hijacking**
  - SESSION IDs and tokens reside in RAM
  - Cookies are stored, and present in backups
  - BAD: sometimes use IP Address as SID (CWE-291)
- Multi-factor authentication may limit exploration
  - Cookie from different IP Address? Invalidate it
  - Cookie from different browser? Invalidate it

Session Hijacking – Sniffing/Interception

- Sessions can be stolen from Cookie repository
  - If device is compromised
  - If they do not expire (CWE-613)

- Browser can be led to provide the Cookie/Token to a **malicious server**
  - Attacker listens for DNS request of http://company.com and provides the address of the malicious server
  - MiTM attacks with non secure (no TLS) services

Session Hijacking – Brute Force

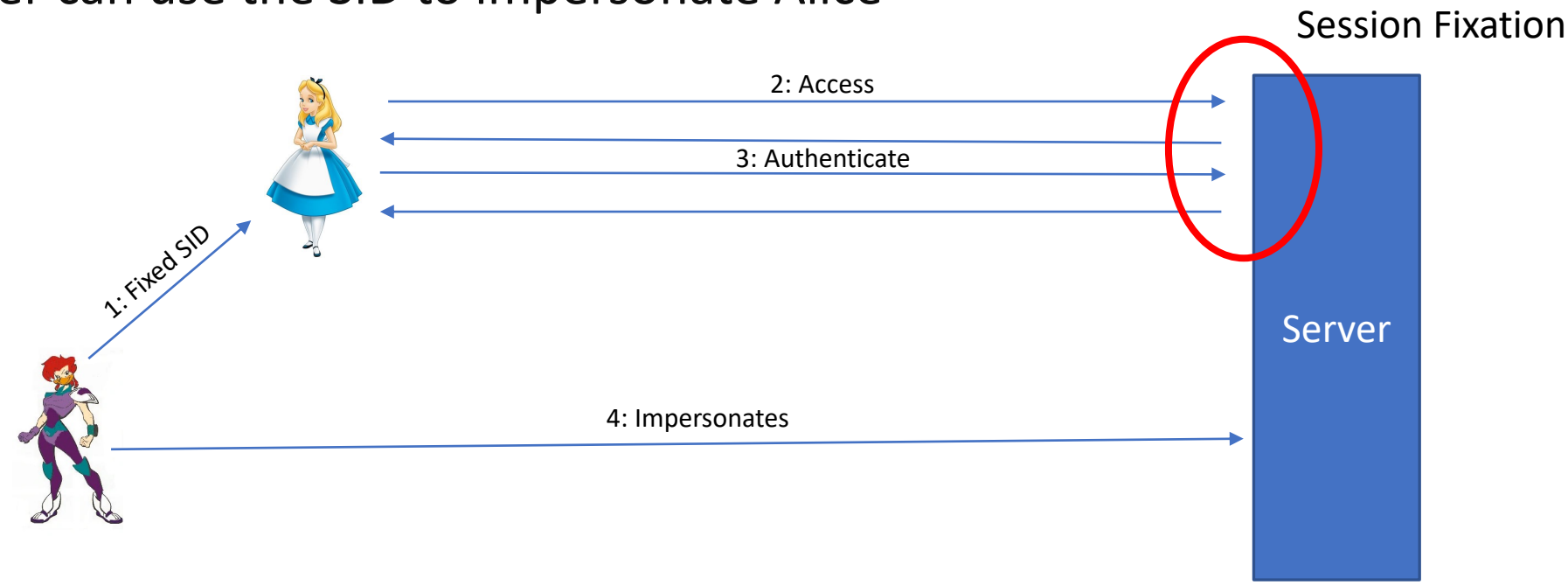
- SIDs and Cookies must have high entropy (CWE-331)
  - Should result from a hash or UUID
  - Caveat: calculating a hash from a timestamp is a bad pattern (CWE-330)
    - Timestamp is predictable
- Otherwise, attacker may **brute force values of active sessions**
  - Send multiple requests with varying SID/Cookies until access is granted
- Same can be done for username/passwords
  - Passwords are weak links
  - User enumeration will reduce the attack time (CWE-200)
  - Applicable to many CPEs

Session Fixation

- SIDs from a non-authentication state **must be invalidated before authorizing a new session** (CWE 384)
  - Alternative is to add a secondary Cookie with a random text
- Attacker may force a predictable SID and wait for authentication
  - SID will be kept after authentication, granting access to the attacker
  - If the attacker can force a known session identifier on a user
    - Once the user authenticates, the attacker has access to the authenticated session
- Detected by observing the Cookie/SID before and after authn

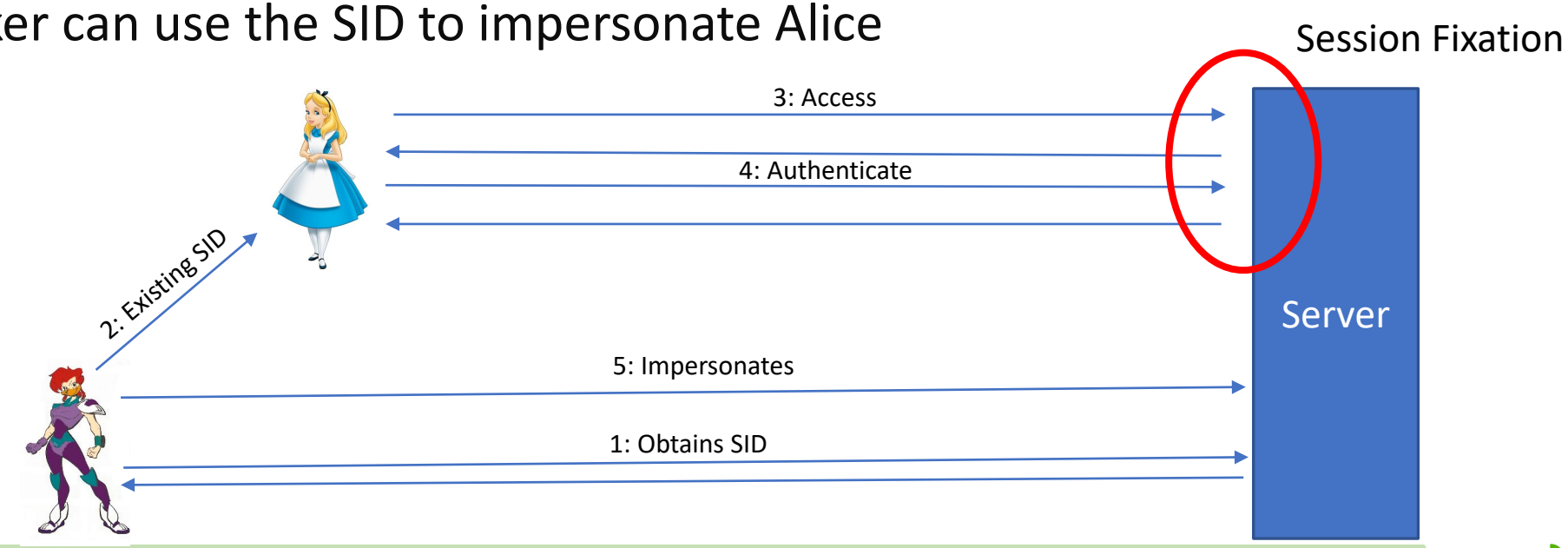
Session Fixation - Scenarios

- Freely controlled SID
  - Attacker says: Hey Alice, check this https://server.com?SID=KNOWN_TO_ATTACKER
  - If Alice accesses the URL and logs on
  - The attacker can use the SID to impersonate Alice



Session Fixation - Scenarios

- Pre-Generated SID
  - Attacker obtains SID from server
  - Attacker says: Hey Alice, check this https://server.com?SID=EXISTING_SID
  - If Alice accesses the URL and logs on
  - The attacker can use the SID to impersonate Alice



Session Fixation - Scenarios

- Cross-Domain Cookie
  - Attacker creates evil.server.com and Alice has account at good.server.com
  - Attacker says: Alice, check this http://evil.server.com and provides a cookie *.server.com
  - If Alice accesses the URL and logs on (The cookie is provided)
  - The attacker can use the Cookie to impersonate Alice