OWASP A2
Broken Authentication

JOÃO PAULO BARRACA

OWASP A2 – Broken Authentication

- Application functions related to **authentication** and **session management** are often implemented incorrectly

- Allow attackers to **compromise passwords, keys, or session tokens**, or to **exploit other implementation flaws**

- Attackers may **assume other users' identities** temporarily or permanently.

OWASP A2 – Broken Authentication

- Prevalence is widespread
  - due to the operation of most identity and access controls.
- Session management is the bedrock of authentication and access controls
  - present in **all stateful applications**
- Attackers can detect broken authentication using **manual means**
- Attackers can exploit them using automated tools
  - **There are extensive password lists and dictionary attack tools**

Changes in the OWASP ranking

- Services evolving from monolithic server applications to microservices
  - Proliferation of HTTP and REST to implement APIs
- Applications are evolving to Progressing Web Applications
  - Single HTML page for entire application
  - Lots of Javascript based logic
  - Resources provided through REST APIs
  - Services exposed to the Internet, used directly by clients
- Impact
  - Logic is moving towards clients
  - State anchors are kept in the clients

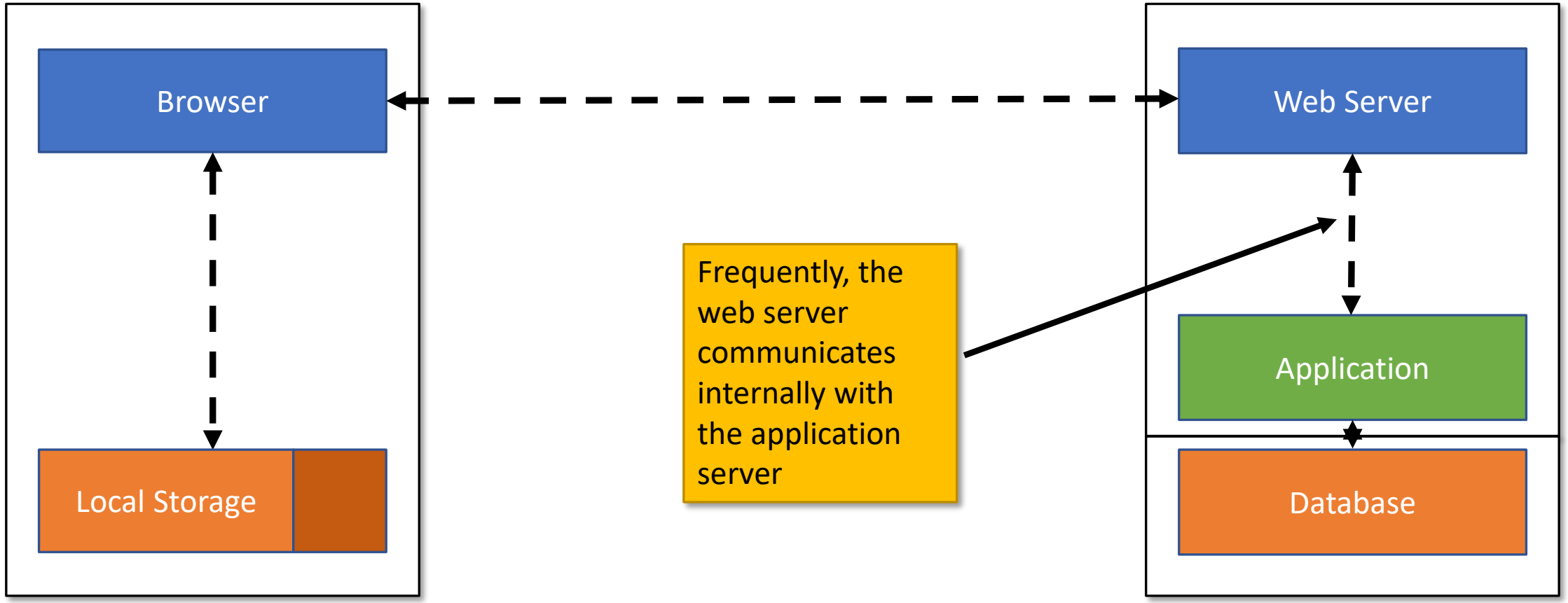
HTTP Basics

The Web

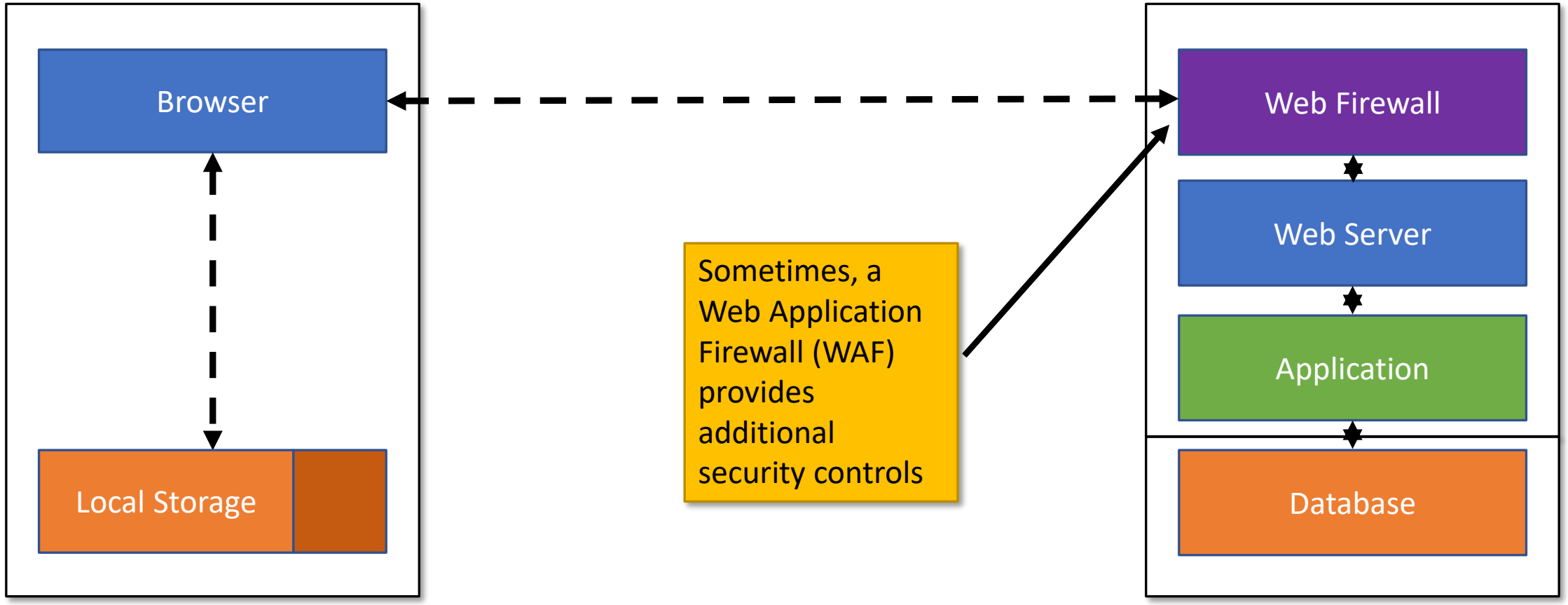
Local storage contains user data inside and outside the browser

Database frequently is on a different host

The Web


Browser
Local Storage
Web Server
Application
Database
Frequently, the web server communicates internally with the application server

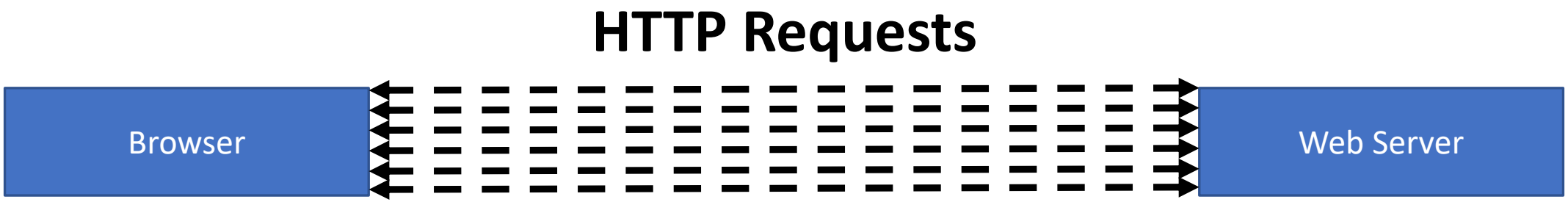
The Web


Browser
Local Storage
Sometimes, a Web Application Firewall (WAF) provides additional security controls
Web Firewall
Web Server
Application
Database

A Web Request

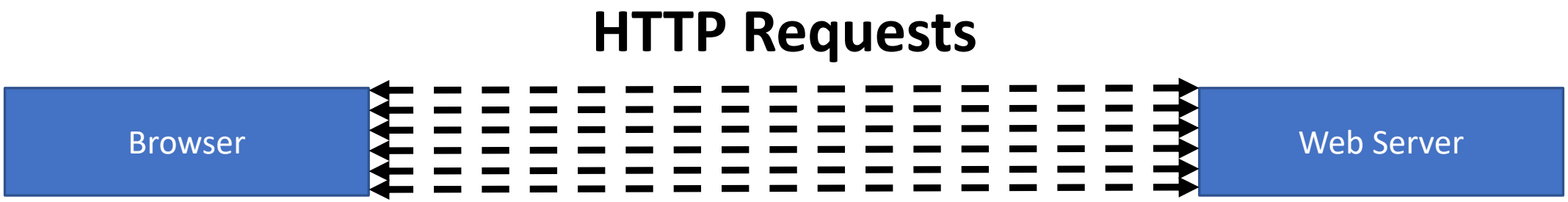
1. Browser asks DNS for the IP address of server.com
2. Browser connects to TCP port 80/443 of server.com
3. Browser sends a request with:
    1. Action: GET, POST, PUT, DELETE
    2. URL: http://server.com
    3. Headers: language, compression, user-agent…

Multiple Web Request



- HTTP is stateless by design
- HTTP Requests are independent of each other
  - Each triggering an individual action
  - Usually **tokens** or **cookies** are included in requests/responses to keep state

HTTP and HTML



- HTTP is not related with HTML
  - You can have HTTP without HTML, and vice versa
- HTTP is a generic transport protocol
  - Usually operated over TCP on port 80 or 8080
- HTML is a language used to define the structure of a web page

HTTP Communication

- HTTP is a standard Client-Server protocol

1. Client establishes a TCP connection with the server on port 80
2. Client sends a HTTP request over that TCP connection
3. Server replies
   - Sends a response
   - HTTP 1.0: Closes the connection
   - HTTP 1.1/2: May keep it *persistent* for some time

- Server only issues replies to requests. It may never contact clients directly

image: https://ruslanspivak.com/

HTTP Communication

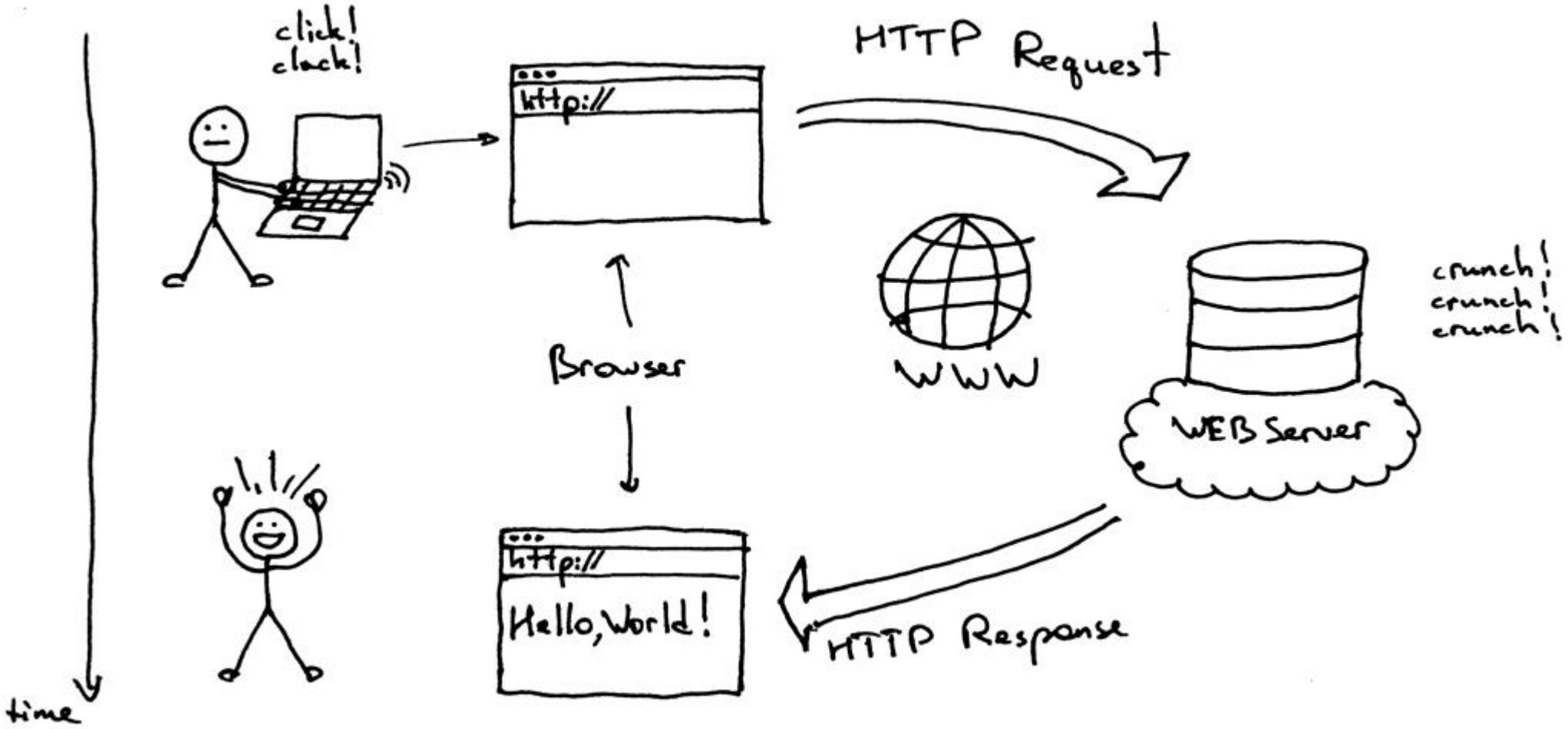
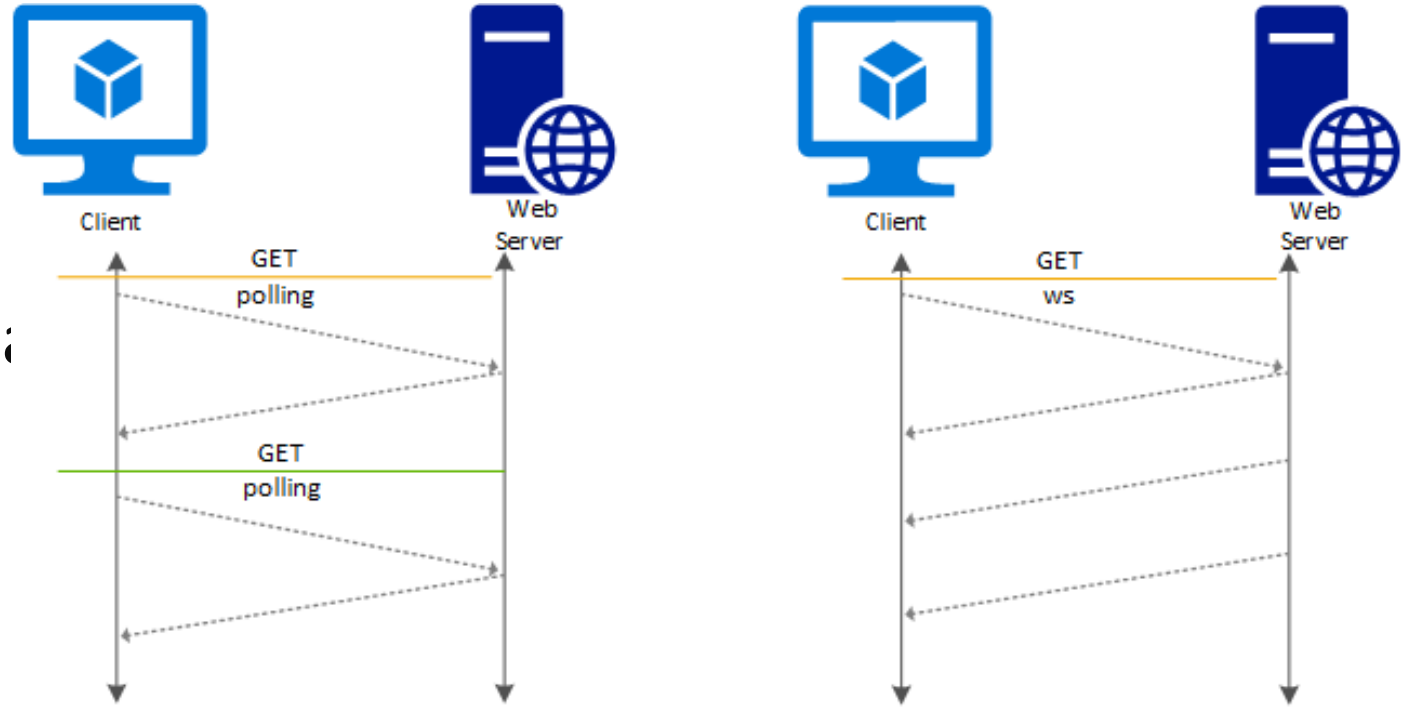


image: Microsoft.com

HTTP Communication

- Actually… servers can contact clients directly with WebSockets
  - Great for low latency asynchronous communications (e.g. VoIP, telemetry)
  - Nightmare for security!
- Client upgrades connection to a WebSocket
- Any participant can send messa
  - No polling is required
    Usually no log is done
  - Client and server must know the message format



HTTP Request

```
$ curl https://elearning.ua.pt  -D - -v


GET / HTTP/1.1
Host: elearning.ua.pt
User-Agent: curl/7.68.0
Accept: */*
```

HTTP Request

HTTP Response

```
$ curl https://elearning.ua.pt  -D - -v

HTTP/1.1 200 OK
Date: Thu, 12 Nov 2020 17:01:16 GMT
Server: Apache
Set-Cookie: MoodleSession=qvnej3ar6u28ndar312jhg1veh; path=/
Expires: Mon, 20 Aug 1969 09:23:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Cache-Control: post-check=0, pre-check=0, no-transform
Last-Modified: Thu, 12 Nov 2020 17:01:16 GMT
Accept-Ranges: none
```

HTTP Response

With server timestamp
Cookie, Cache control

Anything can be a client

```
$ echo -ne 'GET / HTTP/1.1\r\nHost: elearning.ua.pt\r\nUser-Agent: Android
10\r\n\r\n' | ncat --ssl elearning.ua.pt 443

HTTP/1.1 200 OK
Date: Thu, 12 Nov 2020 17:20:12 GMT
Server: Apache
Set-Cookie: MoodleSession=ooma3far88iqh9nvssn598nsuu; path=/
Expires: Mon, 20 Aug 1969 09:23:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Cache-Control: post-check=0, pre-check=0, no-transform
Last-Modified: Thu, 12 Nov 2020 17:20:12 GMT
Accept-Ranges: none
```

Anything can be a client

- Many programs can communicate with HTTP servers
  - A socket is all that is required
- Even Bash can do it

Open TCP to 193.136.173.58:80 into FD5
Write to FD 5
Read from FD 5

```
$ exec 5<>/dev/tcp/193.136.173.58/80
$ echo -e "GET / HTTP/1.1\r\nHost: www.ua.pt\r\n\r\n" >&5
$ cat <&5

HTTP/1.1 301 Moved Permanently
Server: nginx/1.18.0
Date: Thu, 12 Nov 2020 17:26:58 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: https://www.ua.pt/
```

Anything can be a client

- There is no client-side security model
- All parts of a request can be crafted
  - HTTP Headers, Methods, URLs
  - POST content can be manipulated freely
- Control must reside in the server-side context
  - Remember that developers are pushing content to the client? ☺
- There are no input validation processes in the server
  - As long as the HTTP protocol is “generally” observed

Authentication

Authentication - Recap

- Authentication aims to determine the identity of an entity
  - Entity may be user, system, software

- The basic process relies in the verification of some property of the authenticated entity by the authenticator
  - Something that he has
  - Something that he knows
  - Something that he is

Authentication - Recap

- What else can be used?

Authentication - Recap

- Somewhere where he is (location)
- Someone that is close by (neighborhood)
- Something that he has done (past behavior)

- A combination of several
  - 2FA – Two Factor Authentication (e.g. Secret + Cookie)
  - MFA – Multiple Factor Authentication (e.g. Secret + Cookie + Smartphone)

Base HTTP methods

- Makes use of the Authorization header
  - Header is passed to applications as well as user
  - May require password to be in clear text
  - Presents no configurable user interface

- Basic authentication through direct presentation of credentials
  - `Authorization: Basic base64(login:password)`

Base HTTP methods

- Digest authentication
  - Server replies with the authentication arguments in the WWW-Authenticate

```
Authorization: Digest username="Mufasa",
                      realm="testrealm@host.com",
                      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                      uri="/dir/index.html",
                      qop=auth,
                      nc=00000001,
                      cnonce="0a4f113b",
                      response="6629fae49393a05397450978507c4ef1",
                      opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Authentication Flow state

Sessions

- HTTP is stateless and provides no way of keeping state
  - Besides WebSockets in HTML5
- Most applications over HTTP need state for good purposes
  - User preferences
  - Navigation history
  - Authentication state
- Some use it for less noble purposes, usually compromising privacy
  - Track users across multiple sites for advertising purposes
  - Profile user behavior

Flow State keeping mechanisms

- Referer header
- SESSION_ID, or SID or other custom headers
- Cookie
- JSON Web Token

Use of the URL

```
GET /internal/private.html?pass=secret&sid=234234 HTTP/1.1
Host: www.company.com
```

- Input encoded as part of the URL as Request Arguments
- GET request is expected to have side effects
  - Arguments control language, authentication, authorization

Use of the URL

Should be avoided at all cost to transport state

- Arguments are visible in the browser
  - A use problem if your browser is visible: public presentation, remote lecture, over the shoulder eavesdropping
- Arguments may be logged by the webserver
  - Enable compromise if logs are accessed by an attacker
- SEO is broken: different users will have a different URL for the same resource
- Cache may be impacted: unique URLs limit the use of caches

Use of a POST request

```
POST /doLogin HTTP/1.1
Host: company.com
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows 10)
Referer: http://company.com/login
Content-Length: 34

username=john&password=supersecret
```

- URL visible on the browser: http://company.com/doLogin

GET vs POST

- **GET** is used to REQUEST information
  - Can be resent by browsers
  - May be logged, cached, bookmarked, kept in the browser history
  - Should not change server-side state (no side-effects)
    - Frequently it will change state, or create logs
- **POST** is used to UPDATE information
  - Will not be cached, bookmarked, kept in browser history
  - May not be logged
  - Is not visible to users
  - Is expected to change server-side state (has side effects)

Referer Header

- The Referer request header contains the address of the page making the request.

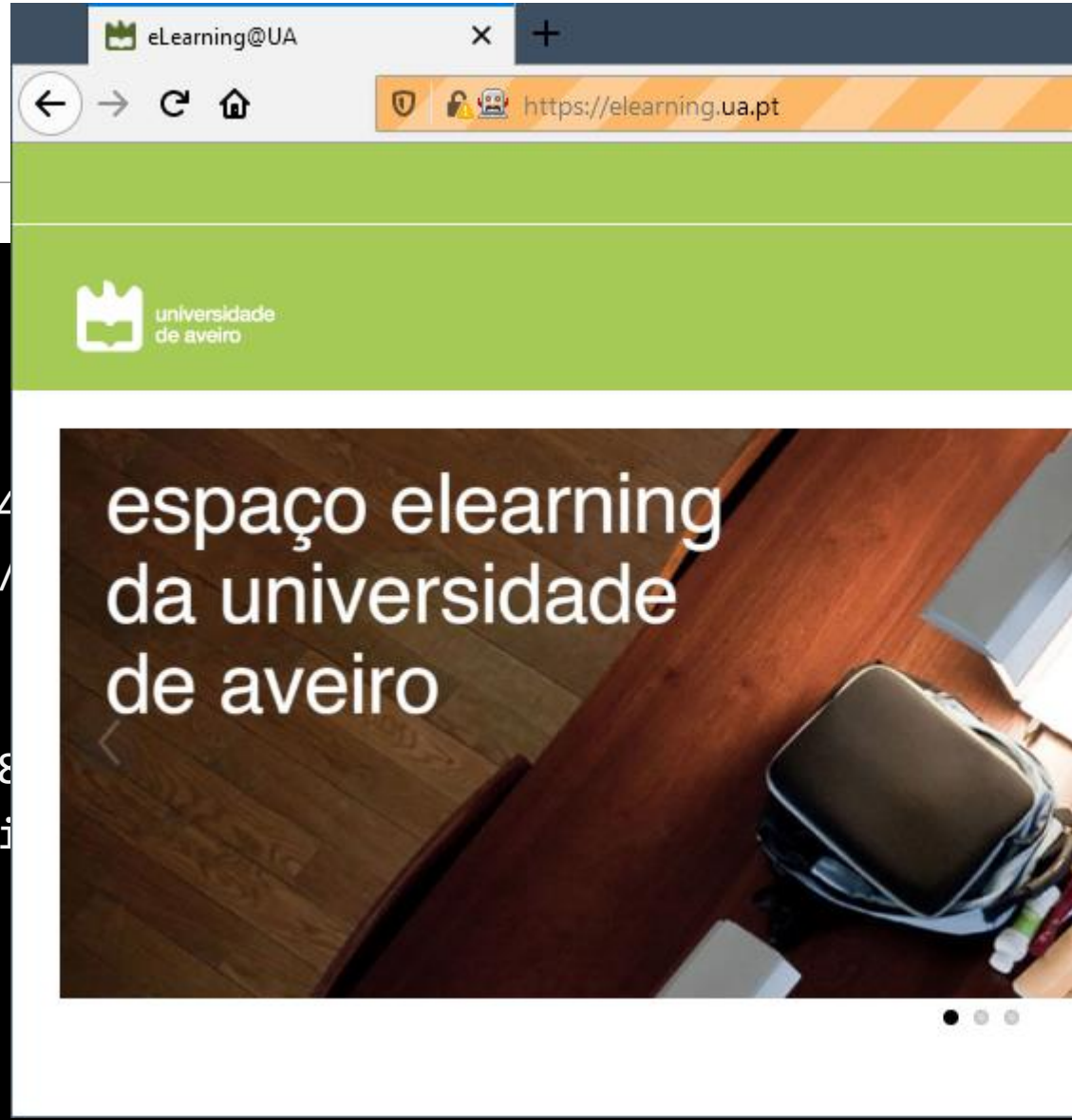
- The Referer header allows servers to identify where people are visiting them from
  - may use that data for analytics, logging, or optimized caching
  - Sometimes used for access control

- Fully user controllable

Referer Header

- First hit: No Referer

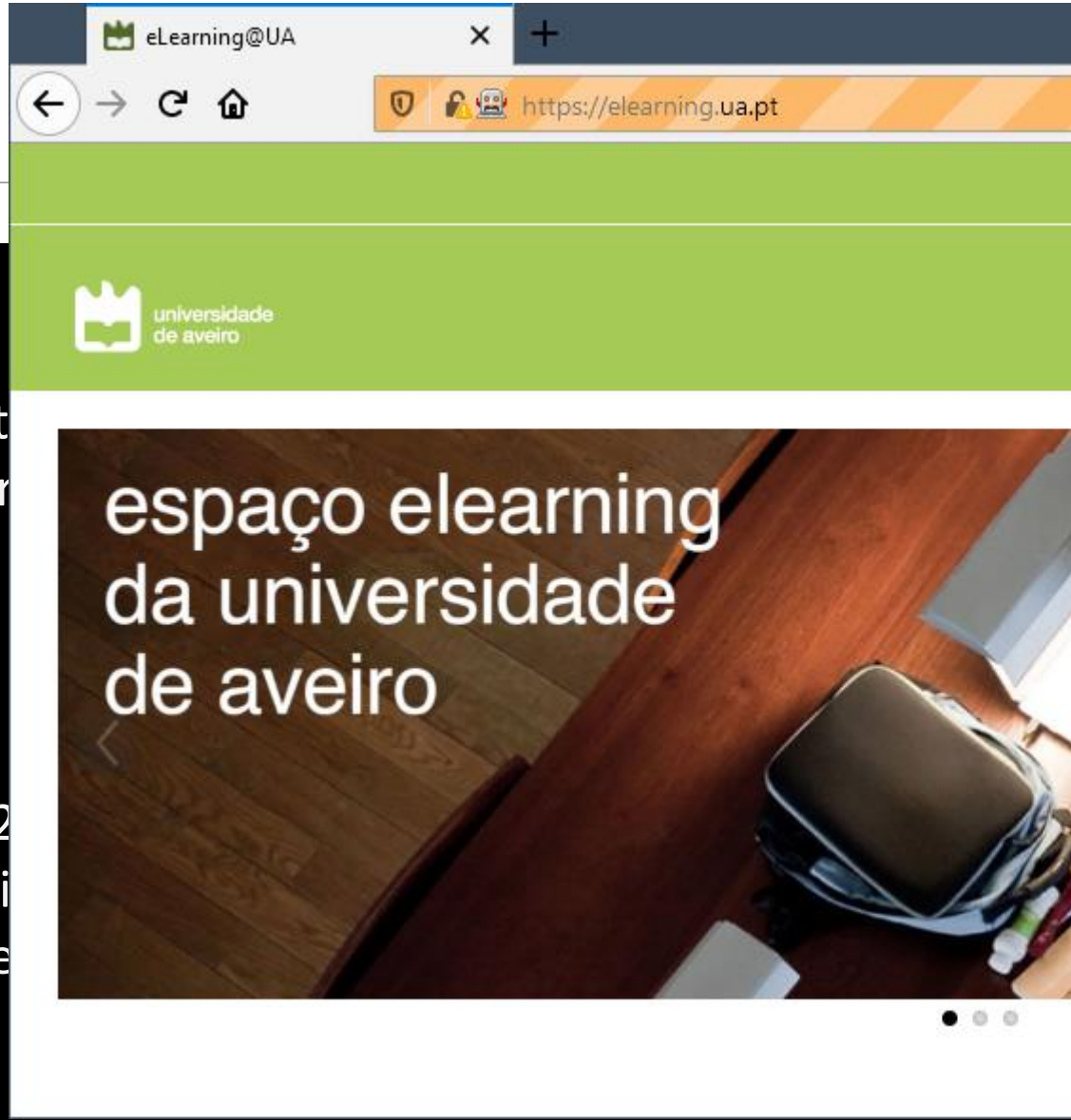
```
GET https://elearning.ua.pt/ HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64
Accept: text/html,application/xhtml+xml,application/
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Cookie: _ga_RWZB1HRVYE=GS1.1.1605202432.1.1.16052028
_gid=GA1.2.1334581424.1605202436; _hjTLDTest=1; _hji
_hjFirstSeen=1; _hjAbsoluteSessionInProgress=0
Upgrade-Insecure-Requests: 1
Host: elearning.ua.pt
```



Referer Header

- Subsequent request

```
GET https://elearning.ua.pt/theme/adaptable/style/print
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; r
Accept: text/css,*/*;q=0.1
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Referer: https://elearning.ua.pt/
Cookie: _ga_RWZB1HRVYE=GS1.1.1605202432.1.1.16052
_gid=GA1.2.1334581424.1605202436; _hjTLDTest=1; _hji
_hjFirstSeen=1; _hjAbsoluteSessionInProgress=0; Moodle
Host: elearning.ua.pt
```



Referer Header

```
GET /internal/private.html HTTP/1.1
Host: www.company.com
Referer: http://www.company.com/loggedin/
```

- Expected meaning: User accessing `/internal/private.html`, and came from `/loggedin` therefore it was authenticated

- In reality
  - Referer header MAY be set by the browser
  - Was meant for origin authentication, is used for authorization
  - Falls in the TOCTOU: Time-of-check time-of-use

SESSION ID

- A value, set by the server in the HTML/Javascript
  - Kept manually by the HTML/JS logic
  - Browser is unaware of it
- URLs in the HTML include the SESSION ID

```
<a href=http://company.com?PHPSESSION=value>resource</a>
```

- SESSION ID added to requests
  - Header or explicit argument in GET actions
  - Header or body in POST actions

Cookies (RFC 6265)

- ASCII text created by the server and sent to the client
  - HTTP Header - Set-Cookie: VALUE
- Stored in the clients' cookie jar
  - A file or simple database
  - The client may freely delete (or edit) cookies
- Client resends the **`Cookie`** header to servers
  - in every request made for which there is a compatible cookie
  - Format is: **`Cookie: VALUE`**

Cookies (RFC 6265)

- Server can keep context using the cookie provided
  1. Receives a Cookie from the client
     1. Cookie can contain the session identifier
  2. Fetches context (session)
  3. Provides a customized answer

- Cookies are used as a token enabling authorization
  - When set as the result of an authentication process
  - Allow obtaining the identity associated with the request

- Loosing a Cookie opens the door to *impersonation*

Cookies (RFC 6265)

- Cookie scope and lifetime are set by the server in the client response

```
Set-Cookie: <nome-cookie>=<valor-cookie>
Set-Cookie: <nome-cookie>=<valor-cookie>; Expires=<date>
Set-Cookie: <nome-cookie>=<valor-cookie>; Max-Age=<non-zero-digit>
Set-Cookie: <nome-cookie>=<valor-cookie>; Domain=<domain-value>
Set-Cookie: <nome-cookie>=<valor-cookie>; Path=<path-value>
Set-Cookie: <nome-cookie>=<valor-cookie>; Secure
Set-Cookie: <nome-cookie>=<valor-cookie>; HttpOnly
Set-Cookie: <nome-cookie>=<valor-cookie>; SameSite=Strict
Set-Cookie: <nome-cookie>=<valor-cookie>; SameSite=Lax
```

Cookies (RFC 6265)

- Client -> Server
  No cookie sent

- Server -> Client

```
Set-Cookie: MoodleSession=0r6mroovg98o338clahfd177g0; path=/
```

- Client -> Server

```
Cookie: MoodleSession=0r6mroovg98o338clahfd177g0
```

JWT - JSON Web Tokens

- Concatenation of 3 texts
  - base64(header) + '.' + base64(payload) + '.' + base64(signature)
  - signed with a HMAC or Asymmetric crypto (RSA)

```
header = { "alg" : "HS256",  "typ" : "JWT" }

payload = {"loggedInAs" : "admin","iat" : 1422779638}

signature=HMAC-SHA256(secret,base64(header)+'.'+base64(payload))
```

JWT - JSON Web Tokens

- Provide mechanisms for token refresh, limiting impact due to a lost token
- Access Token – JWT Token that authorizes the user – very limited lifespan
  - Is used in every request and has higher exposition
- Refresh Token – Random Token only to refresh Access Token
  - Only used to refresh the Access Token
  - Longer lifetime
- After all tokens expire, the authentication process must be restarted

```
payload = {"loggedInAs" : "admin","iat" : 1422779638}
```

Exploitation and Prevention

What makes an application vulnerable?

- Credentials can be guessed or overwritten as a result 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)

- 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 authn 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

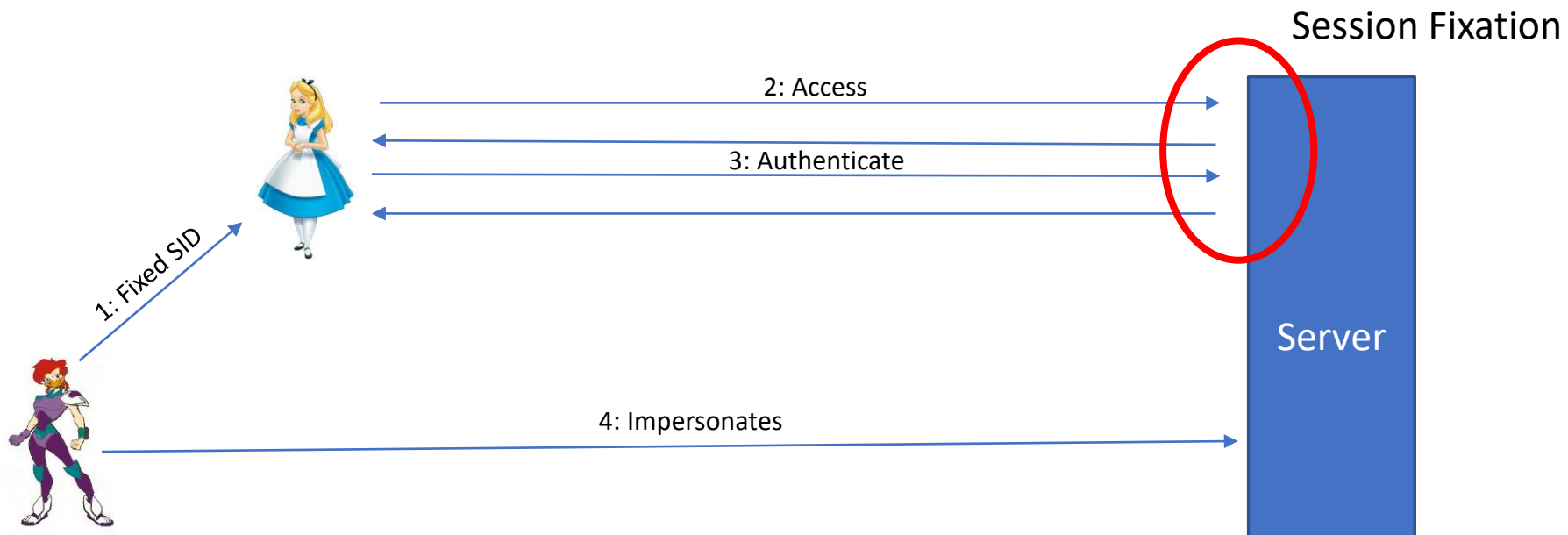
- SIDs from a non-authentication state must be invalidated before authorizing a new session (CWE 384)
  - Alternative is to add a secondary Cookie with a random text

- Attacker may force a predictable SID and wait for authentication
  - SID will be kept after authentication, granting access to the attacker
  - Hey Alice, check this https://server.com?SID=KNOWN_TO_ATTACKER

- Detected by observing the Cookie/SID before and after authn

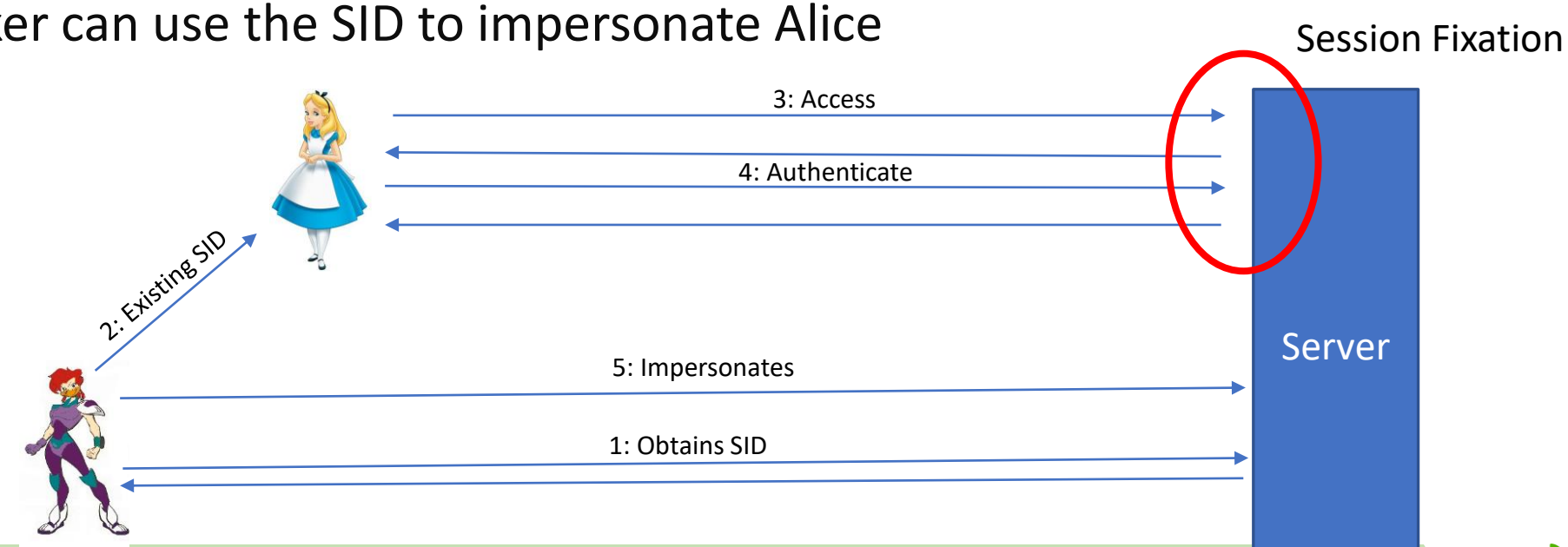
Session Fixation - Scenarios

- 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