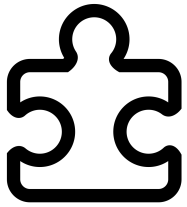
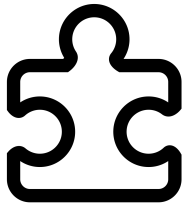


SOP Quiz Solution

Recall that a same-origin policy requires requests to access data must be from the same origin.

What is the definition of an origin?

A combination of URI (UniformResource Identifier) scheme, hostname, and port number.



SOP Quiz Solution

```
https://example.org/absolute/URI/with/absolute/path/to/resource.txt
//example.org/scheme-relative/URI/with/absolute/path/to/resource.txt
/relative/URI/with/absolute/path/to/resource.txt
relative/path/to/resource.txt
../../../resource.txt
./resource.txt#frag01
resource.txt
#frag01
```

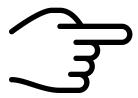
SOP Review

Same Origin Policy (SOP) for DOM:

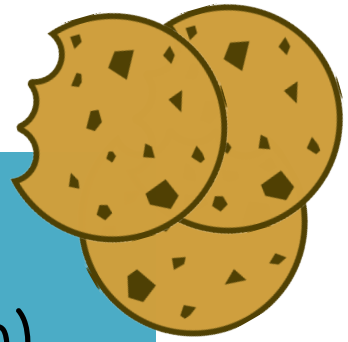
-Origin A can access origin B's DOM if A and B have same (protocol, domain, port)

SOP Review

This lesson:



Same Origin Policy (SOP) for cookies:
Based on ([scheme], domain, path)

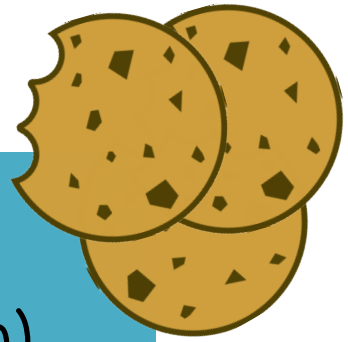


(scheme://domain:port/path?params)

SOP Review

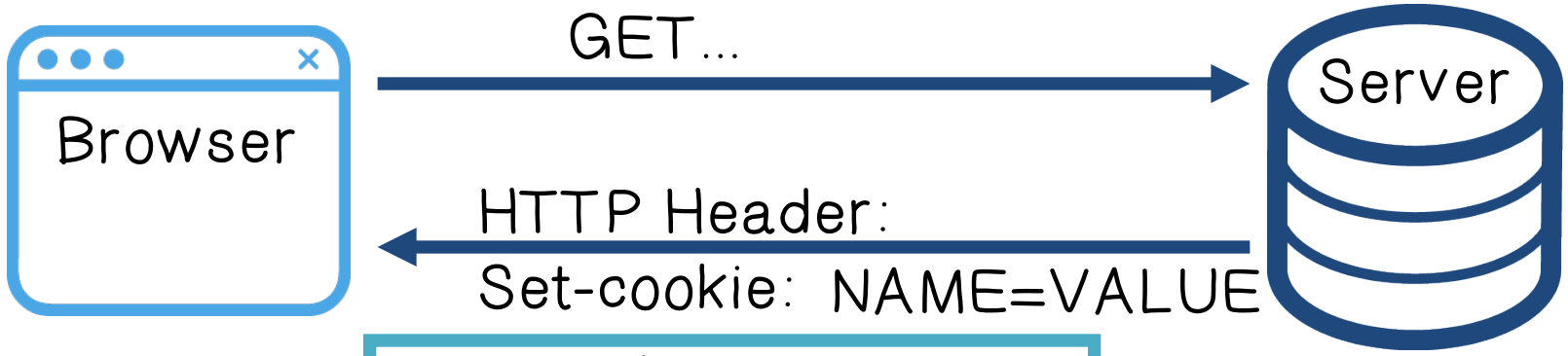
This lesson:

Same Origin Policy (SOP) for cookies:
Based on ([scheme], domain, path)



Optional

SOP and Cookies



scope

domain=(when to send);
path=(when to send)
secure=(only send over SSL)
expires=(when expires)
HttpOnly ← weak XSS defense
SameSite=[lax | strict] ← weak CSRF defense

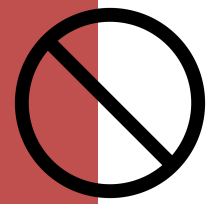
Default scope is domain and path of setting URL

Scope Setting Rules

Domain: any domain-suffix of URL-hostname, except TLD

- login.site.com can set cookies for all of .site.com but not for another site or TLD

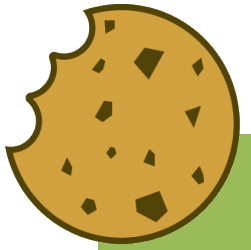
Allowed Domains:	Disallowed Domains:
login.site.com	other.site.com
.site.com	othersite.com
	.com



- Path: can be set to anything

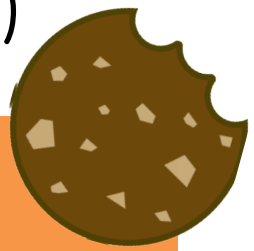
Setting and Deleting Cookies by Server

Cookies are identified by (name, domain, path)



Cookie1

name= userid
value= test
domain= login.site.com
path= /
secure

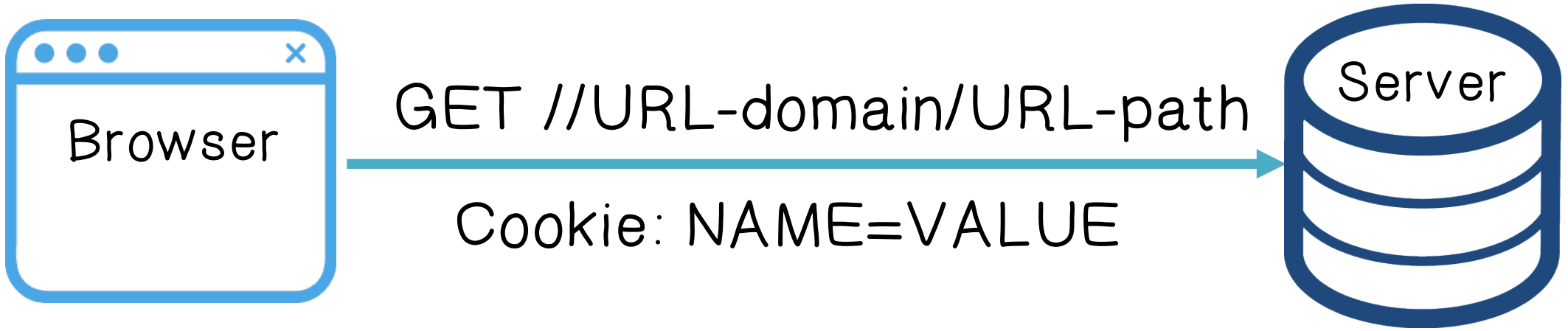


Cookie2

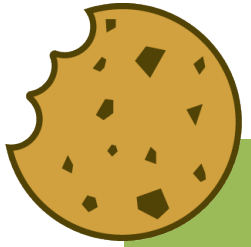
name= userid
value= test123
domain= .site.com
path= /
secure

distinct cookies

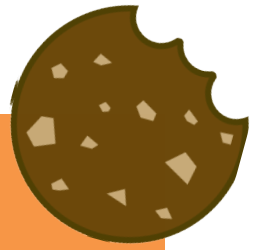
Reading Cookies on the Server



Goal: server only sees cookies in its scope



Reading Cookies on the Server



Cookie1

name= userid
value= u1
domain= login.site.com
path= /
secure

Cookie2

name= userid
value= u2
domain= .site.com
path= /
non-secure

http://checkout.site.com/	cookie: userid=u2
http://login.site.com/	cookie: userid=u2
https://login.site.com/	cookie: userid=u1; userid=u2

Client-side read/write: document.cookie

Setting a cookie in Javascript:

document.cookie = “name=value; expires=...; ”

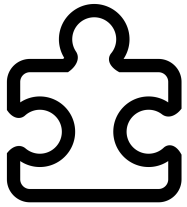
Reading a cookie:

alert(document.cookie) → prints string containing all cookies available for document (based on [protocol], domain, path)

Deleting a cookie:

document.cookie = “name=; expires= Thu, 01-Jan-70”

HttpOnly cookies: not included in document.cookie and cannot be accessed by client-side scripts.

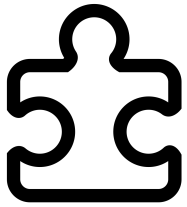


SOP Security Quiz Solution

Given this website: “http://www.example.com/dir/page.html”

Determine the outcome (success or failure) for each compared URL. Check each URL that has the same origin as the given site.

- [x] http://www.example.com/dir2/other.html
- [x] http://www.example.com/dir/page2.html
- [x] http://username:password@www.example.com/dir2/other.html
- [ ] http://www.example.com:81/dir/other.html
- [ ] http://example.com/dir/other.html
- [ ] https://www.example.com/dir/other.html



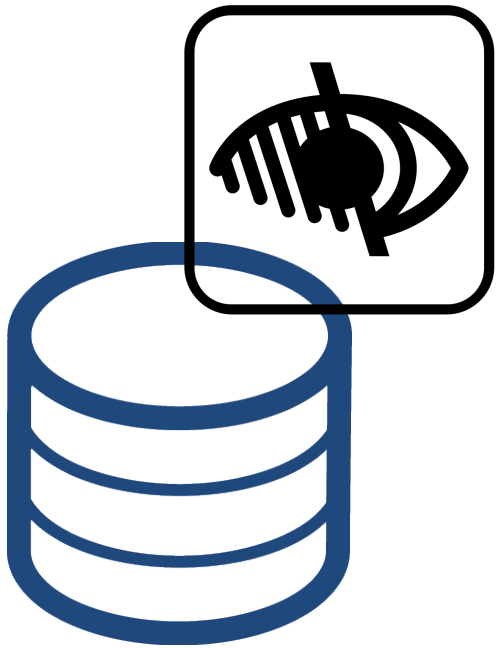
Fill in the blanks with the most correct answer.

Answer choices: session, persistent, secure, HttpOnly, SameSite, Third-party, Super, Zombie

Super	A cookie with an origin of a top-level domain
Zombie	A cookie that is regenerated after it is deleted
SameSite	A cookie that can only be sent in requests originating from the same origin as the target domain.
HttpOnly	A cookie that cannot be accessed by client-side APIs.
Third-party	A cookie that belongs to a domain that is different than the one shown in the address bar.
Session	An in-memory cookie. It does not have an expiration date. It is deleted when the browser is closed.
Persistent	A cookie that has an expiration date or time. Also called tracking cookies.
Secure	A cookie that can only be transmitted over an encrypted connection.

Cookie Protocol Problems

The server is blind



- Does not see cookie attributes (e.g. secure, HttpOnly)
- Does not see which domain set the cookie
- Server only sees:
  Cookie: NAME=VALUE

Cookie Protocol Problems
Example 1

1. Alice logs in at login.site.com → session-id of Alice's session
2. Alice visits evil.site.com → session-id of Badguy's session
3. Alice visits course.site.com → thinks it is from badguy

Cookie Protocol Problems
Example 1

Problem: course.site.com expects session-id from login.site.com; cannot tell that session-id cookie was overwritten

Cookie Protocol Problems
Example 2

Alice logs in at **https**://accounts.google.com

```
set-cookie: SSID=A7_ESAgDpKyk5TGnf; Domain=.google.com; Path=/ ;
                    Expires=Wed, 09-Mar-2026 18:35:11 GMT;
                    Secure; HttpOnly
set-cookie: SAPISID=wj1gYKLFy=RmWybP/ANtKMtPIHNambvd14;
Domain=.google.com;Path=/ :
Expires=Wed, 09-Mar-2026 18:35:11 GMT; Secure
```

Alice visits **http**://www.google.com (cleartext)

Network attacker can inject into response

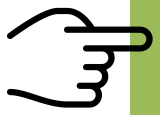
Set-Cookie: SSID=badguy; secure

Which will secure and overwrite secure code

Cookie Protocol Problems
Example 1

Problem: Network attacker can intercept and re-write HTTPS cookies! HTTPS cookie value cannot be trusted.

Interaction with the DOM SOP



Path separation is done for efficiency not security:
x.com/A is only sent the cookies it needs

Interaction with the DOM SOP

- Cookie SOP path separation:
  x.com/A does not see cookies of x.com/B

- Not a security measure because x.com/A still has access to DOM of x.com/B, for example using:

```
<iframe src="x.com/B"></iframe>
alert(frames[0].document.cookie);
```

Cookie Protocol Problems

Cookies have no integrity!

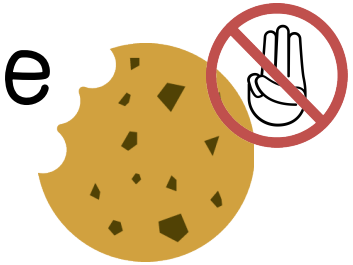
Cookie Protocol Problems

Cookies have no integrity!

User can change and delete cookie values

- Edit cookie database (FireFox: cookies.sqlite)
- Modify Cookie header (FireFox: TamperData extension)

Example: Shopping cart software



User edits cookie file
(cookie poisoning)

Set-cookie:
shopping cart total= $150

Cookie:
shopping cart total= $15

Similar problem with hidden fields:

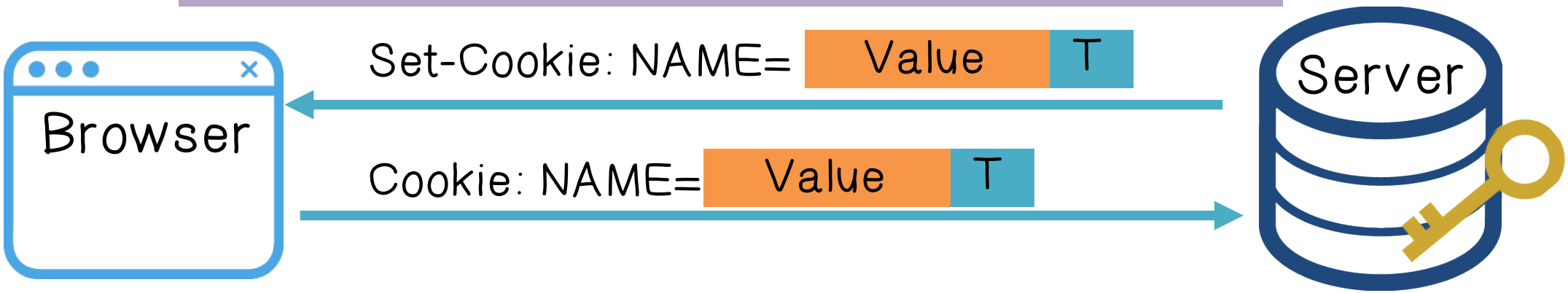
```
<INPUT TYPE="hidden" NAME=price
VALUE="150">
```

Cryptographic Checksums

Goal: data integrity

- Requires server-side secret key k unknown to browser

Generate tag: T MACsign(k, SID || name || value)



Verify tag: MACverify(k, SID || name || value, T)

Cryptographic Checksums

Example: ASP.NET

System.Web.Configuration.MachineKey

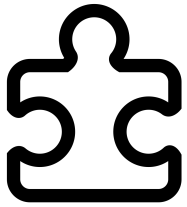
- Secret web server key intended for cookie protection

Creating an encrypted cookie with integrity

- HttpCookie cookie= new HttpCookie(name, val);
  HttpCookie encodedCookie= HttpSecureCookie.Encode (cookie)

Decrypting and validating an encrypted cookie:

- HttpSecureCookie.Decode (cookie)



Checksum Quiz Solution

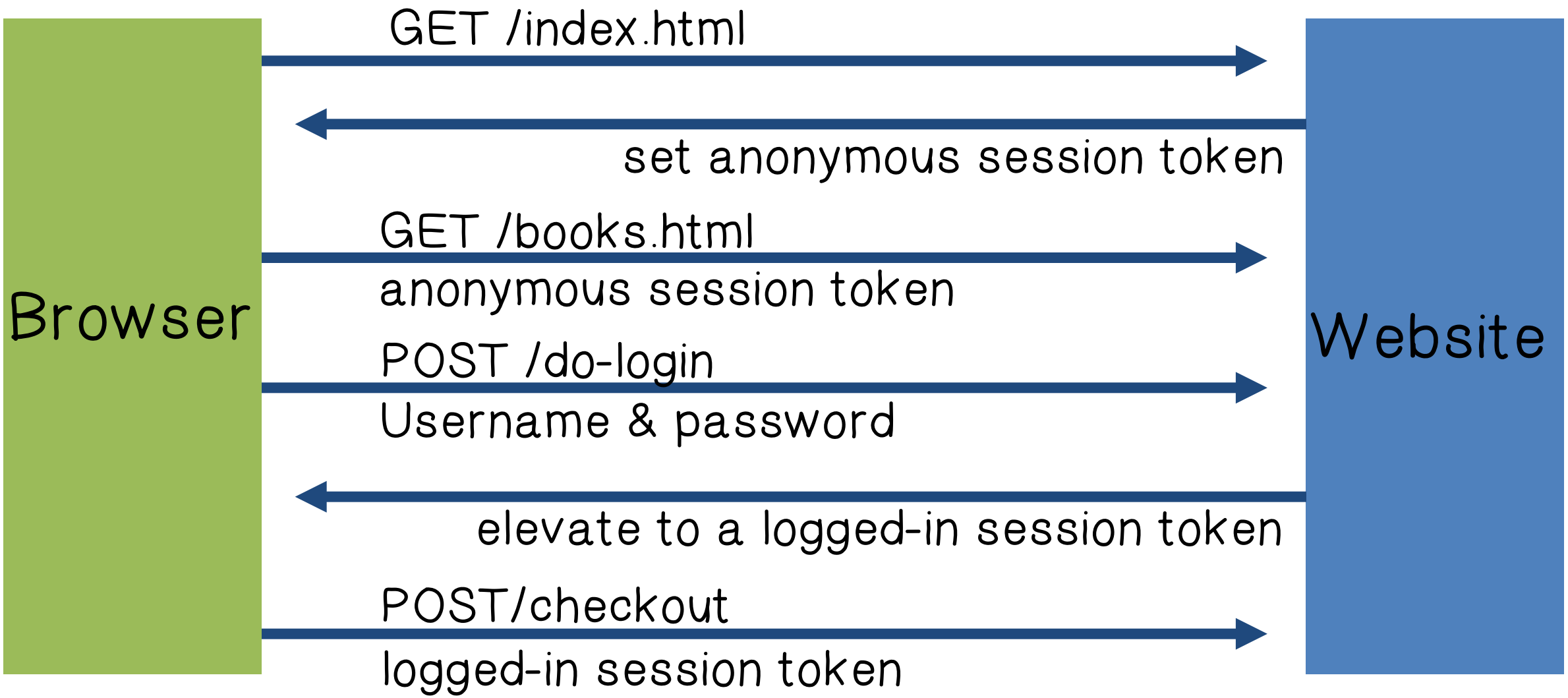
Check all the statements that are true:

- [x] Cryptographic hash functions that are not one-way are vulnerable to preimage attacks
- [ ] A difficult hash function is one that takes a long time to calculate
- [x] A good cryptographic hash function should employ an avalanche effect

Sessions

- A sequence of requests and responses from one browser to one (or more) sites
  - Session can be long (e.g., Gmail) or short
  - Without session management, users would constantly re-authenticate
- Session management: authorize user once; all subsequent requests are tied to user

Session Tokens



Storing Session Tokens

Browser cookie:

- Set-Cookie: SessionToken=fduhye63sfdb

Embed in all URL links:

- https://site.com/checkout ? SessionToken=kh7y3b

In a hidden form field:

- <input type="hidden" name="sessionid" value="kh7y3b">

Storing Session Tokens

Best Method: a combination of all 3:

- Browser cookie, embed in URL, hidden form field

The HTTP Referer Header

Shows the page you are coming from- your referer

```
Host     slogout.espncricinfo.com
User-Agent  Mozilla/5.0 (Windows NT  6.1; rv:5.0) Gecko/20100101
Firefox/5.0
Accept  text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language en-us,en;q=0.5
Accept-Encoding gzip, deflate
Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection keep-alive
Referer http://slogout.espncricinfo.com/index.php?page=index.php?page
=index&level=login
```

The HTTP Referer Header

Problem:

Referer leaks URL session token to 3rd parties

Solution: Referer Suppression

not sent when HTTP site refers to an HTTP site

in HTML5: <a rel="noreferrer" href=www.example.com>

Session Token Security- Logout Procedure

Web sites must provide a logout function:

Let user login as different user.

Prevent others from abusing content

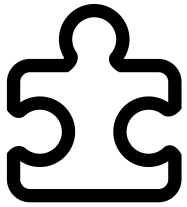
Session Token Security- Logout Procedure

What happens during a logout:

1. Delete SessionToken from client
2. Mark session token as expired on server

! Problem: Many web sites do 1 but not 2!

Especially risky for sites who fall back to HTTP after login

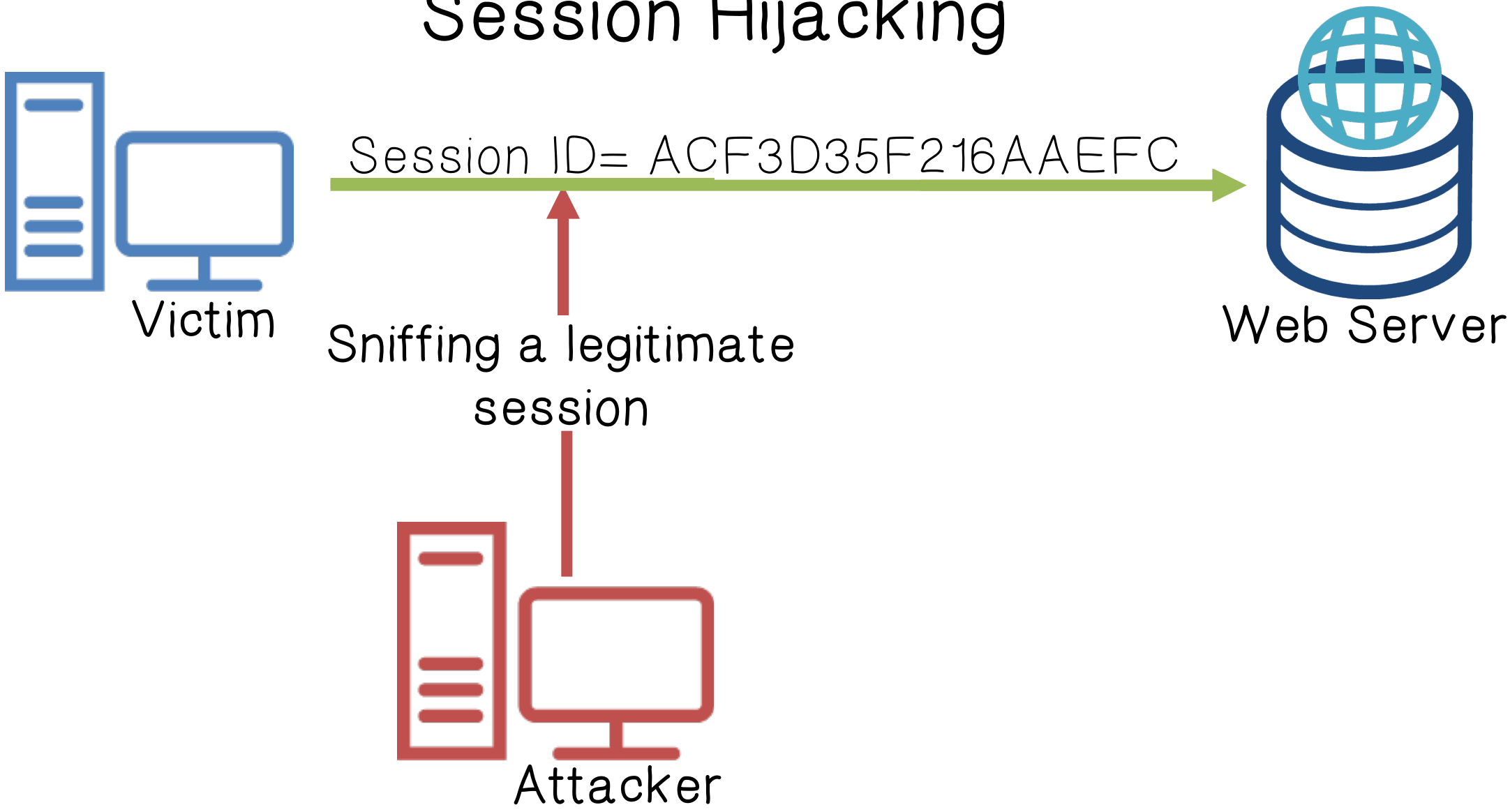


Session Token Quiz Solution

Check all the statements that are true:

- [x] The token must be stored somewhere
- [x] Tokens expire, but there should still be mechanisms to revoke them if necessary
- [ ] Token size, like cookie size, is not a concern

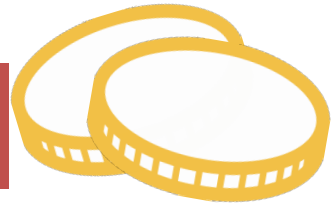
Session Hijacking



Session Hijacking

Session Hijacking

Beware of predictable tokens!



Example 1: Counter:

User logs in, gets counter value, can view sessions of other users

Example 2: Weak MAC token:

Weak MAC exposes secret key from a few cookies, gets counter value, can view sessions of other users

Apache Tomcat: generateSessionId()

Returns random session ID

[server retrieves client state based on session-id]

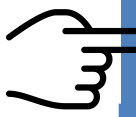
Session Hijacking

Session tokens must be unpredictable to attacker

To generate: Use underlying framework (e.g., ASP, Tomcat, Rails)

Rails: Token=MD5(current time, random nonce)

Session Token Theft



Example 1: Login over HTTPS, but subsequent HTTP

- Enables cookie theft at wireless café (e.g., Firesheep)
- Other ways network attacker can steal token:
  - Site has mixed HTTPS/HTTP pages- token sent over HTTP
  - Man-in-the-middle attacks on SSL

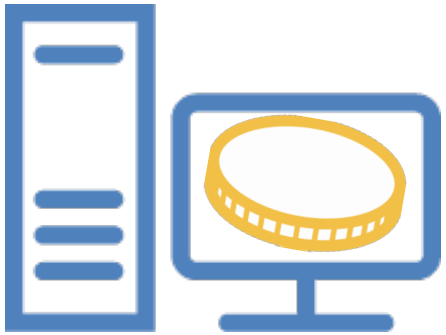
Session Token Theft

Example 2: Cross Site Scripting (XSS) exploits:

- Amplified by poor logout procedures
  - Logout must invalidate token on server

Session Hijacking

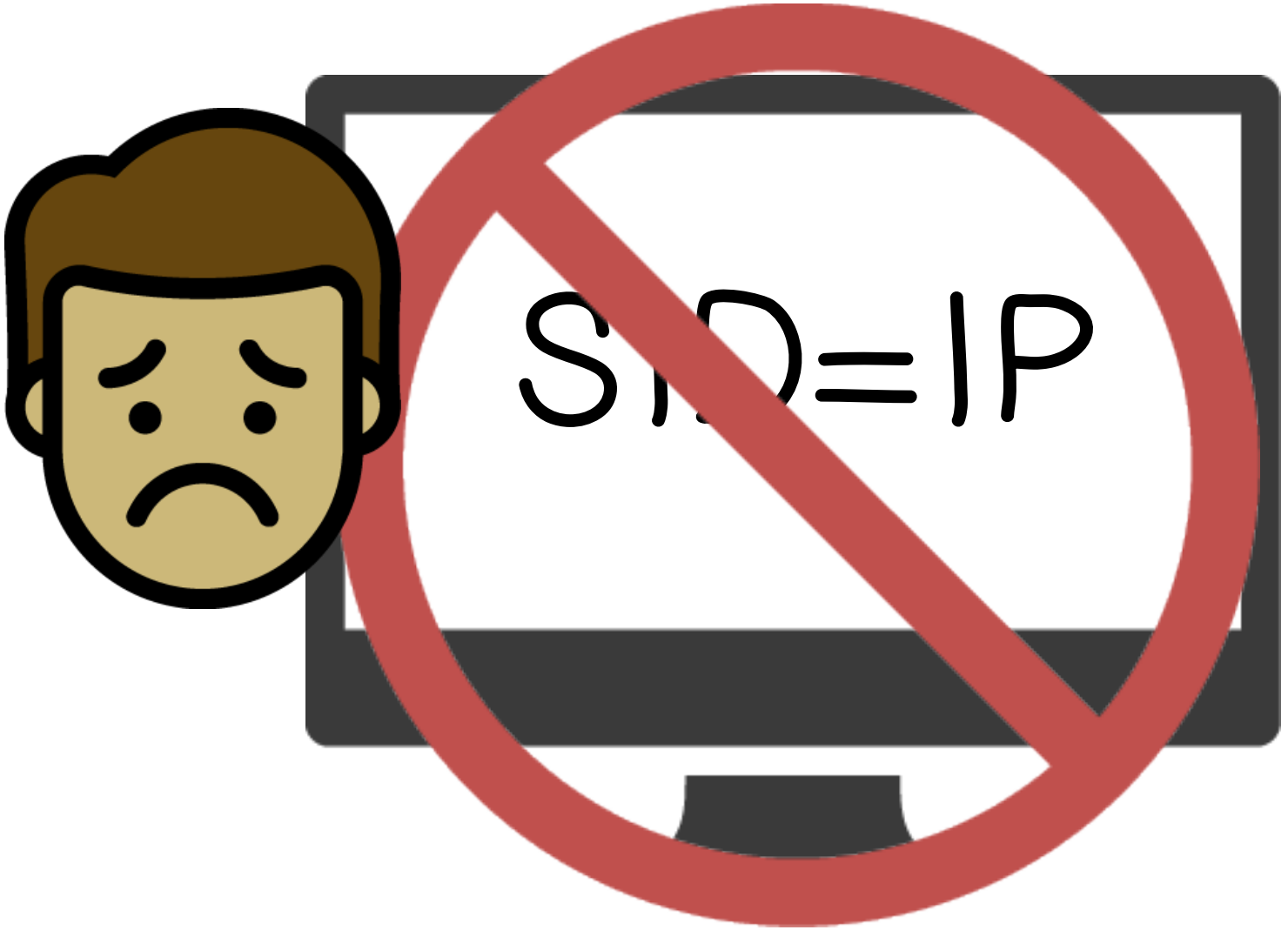
Binding SessionToken to client's computer



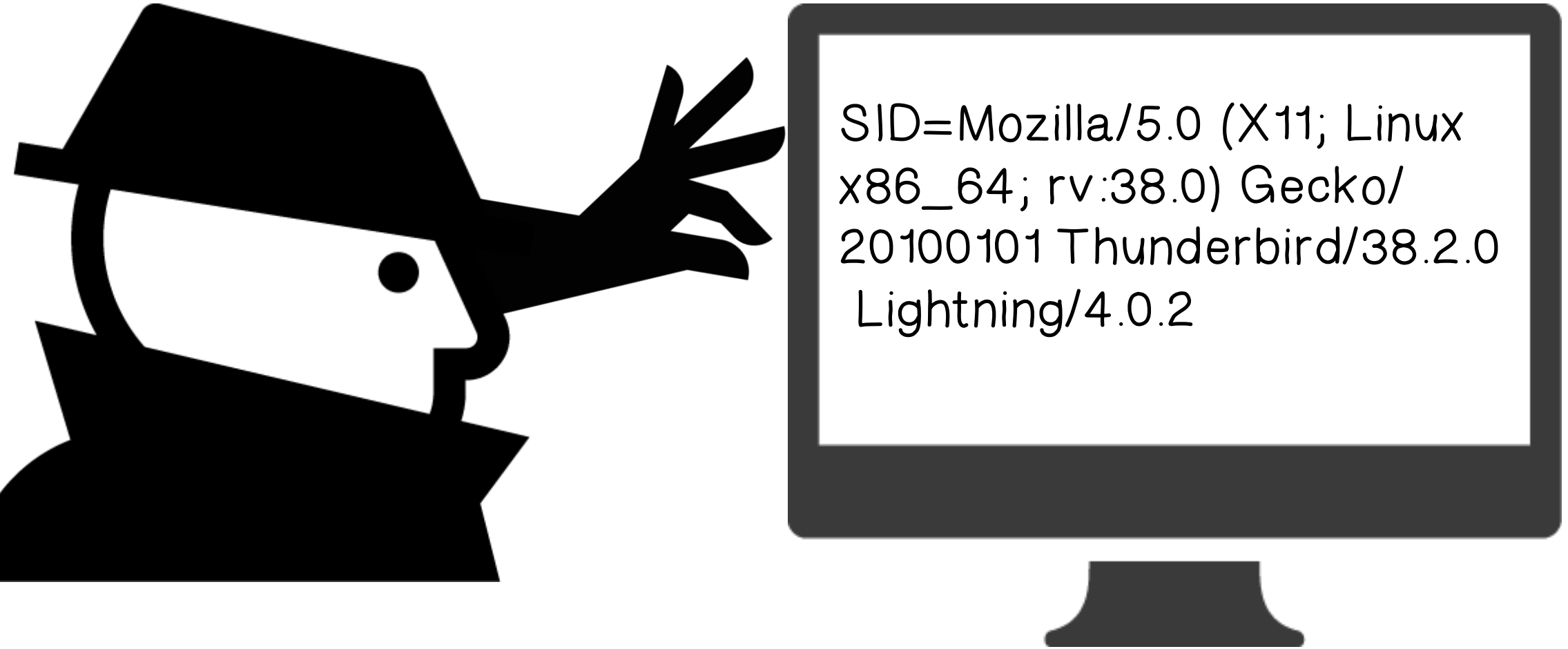
A common idea: embed machine specific data in SID

Session Hijacking

Session Hijacking



Session Hijacking



Session Fixation Attacks

☞ Suppose attacker can set the user's session token:

For URL token, trick user into clicking URL

For cookie tokens, set using XSS exploits

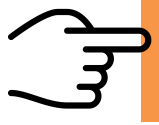
Session Fixation Attacks

Attack: (say, using URL tokens)

1. Attacker gets anonymous session token for site.com
2. Sends URL to user with attacker's session token
3. User clicks on URL and logs into site.com
4. Attacker uses elevated token to hijack user's session

Session Fixation: Lesson

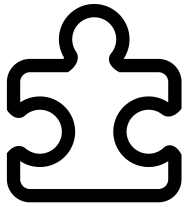
When elevating user from anonymous to logged-in:



always issue a new session token

After login, token changes to value unknown to attacker

- Attacker's token is not elevated



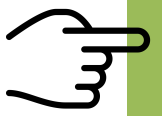
Session Hijacking Quiz Solution

Check all the statements that are true:

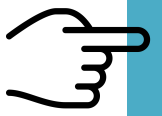
- [x] Active session hijacking involves disconnecting the user from the server once that user is logged on. Social engineering is required to perform this type of hijacking.

- [ ] In Passive session hijacking the attacker silently captures the credentials of a user. Social engineering is required to perform this type of hijacking.

Session Management Summary

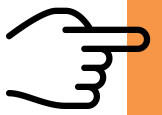


Always assume cookie data retrieved from client is adversarial



Session tokens are split across multiple client state mechanisms.

- Cookies, hidden form fields, URL parameters
- Cookies by themselves are insecure (CSRF, cookie overwrite)
- Session tokens must be unpredictable and resist theft



Ensure logout invalidates session on server