CSCI-UA.9480
Introduction to Computer Security

Session 4.1
Browser Security Model

Prof. Nadim Kobeissi

Defining Browser Security Goals

Also before we start: Practical Assignment 2 is now online.

4.1a

Browser security goals.

- *Confidentiality*: information on your device is not put at risk simply by browsing the web.
- *Integrity:* Different websites are managed through different sessions in isolation.

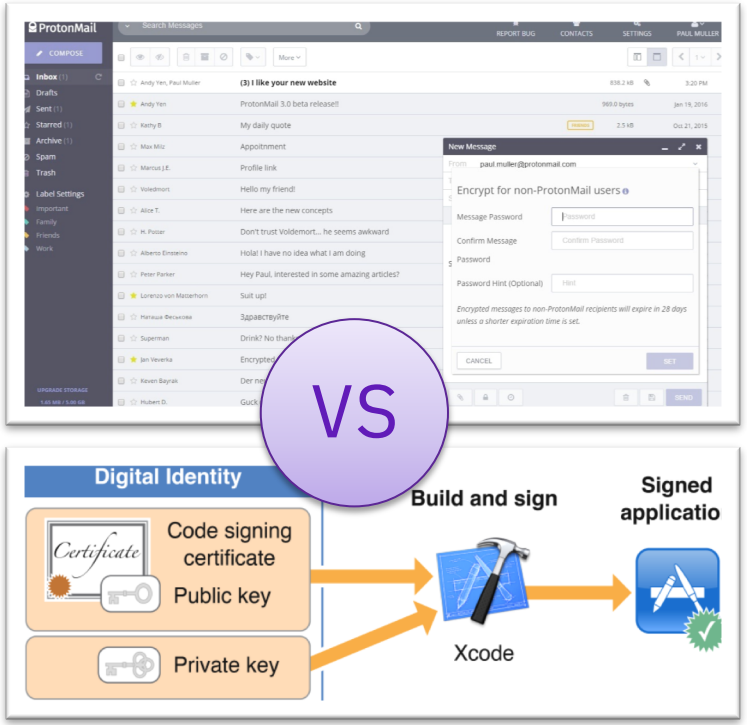
More far-reaching goals could include imbuing web applications with the same security as desktop applications.

Can web apps be as trustworthy as local apps?

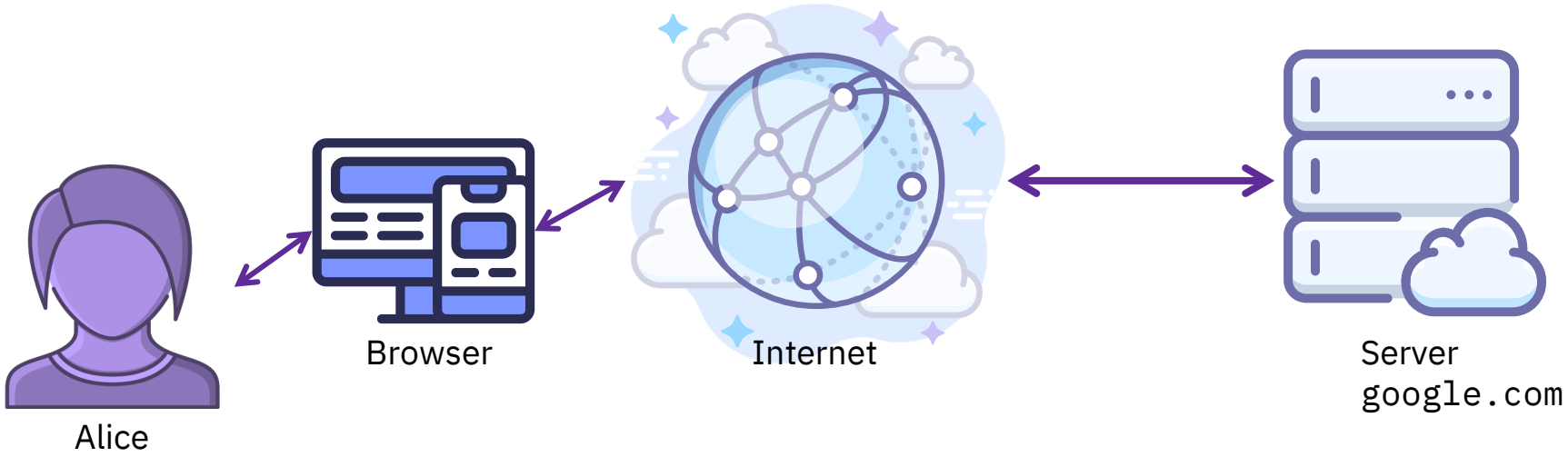
Example: ProtonMail.

- ProtonMail ensures security guarantees through TLS and client-side encryption.
- However, a malicious ProtonMail host server can imperceptibly modify code for select sessions.

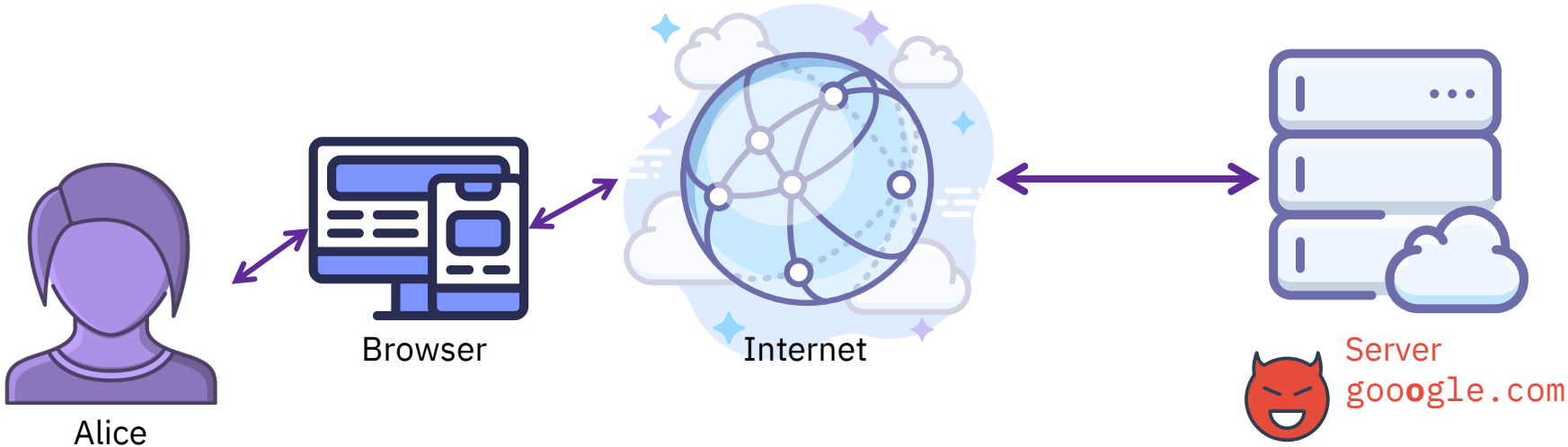
This is very different from desktop and mobile applications, which have signing and versioning.



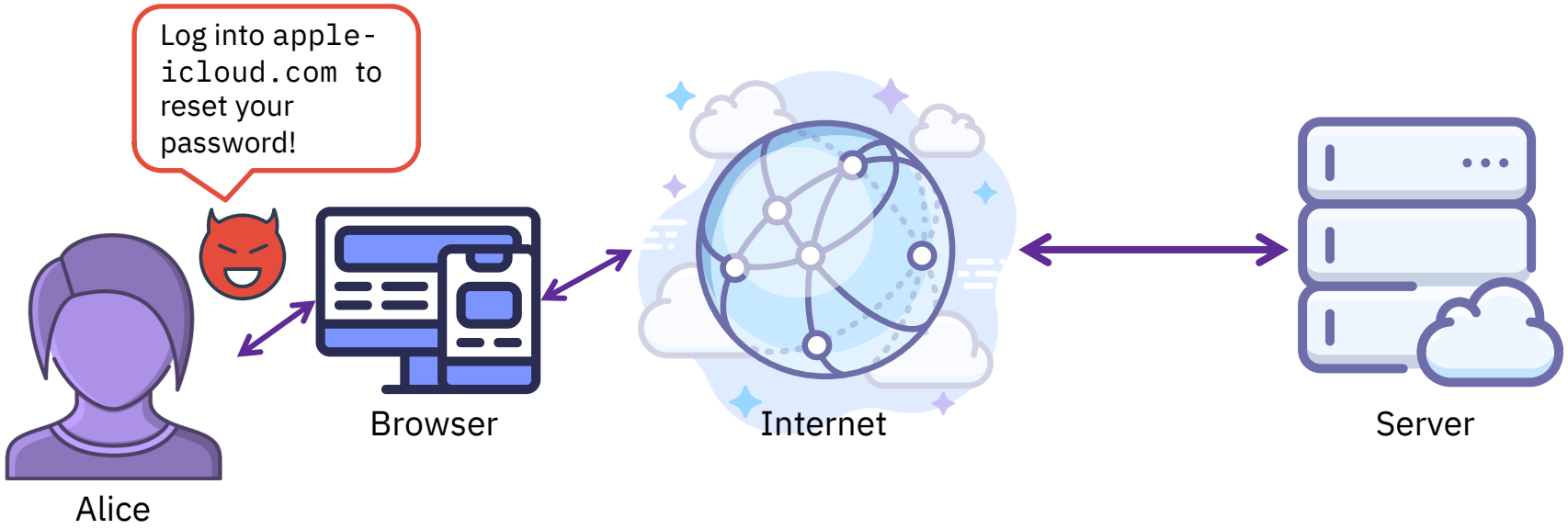
Browser security threat model.



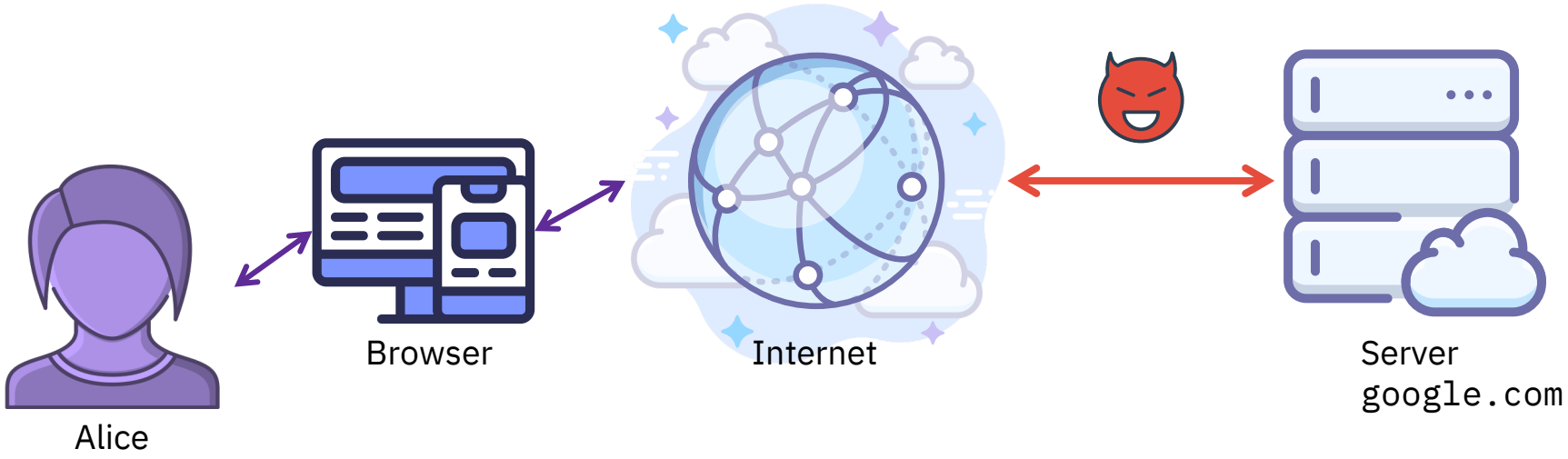
Browser security threats: web attacker.



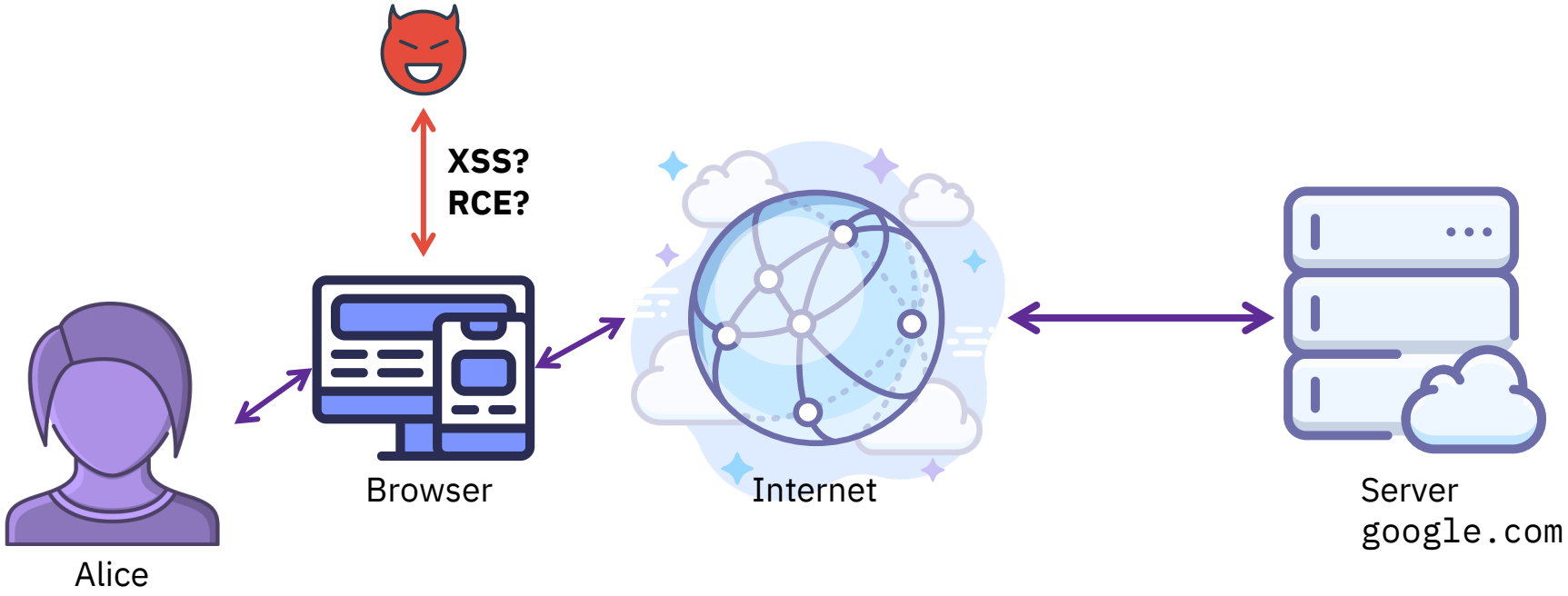
Browser security threats: web attacker.



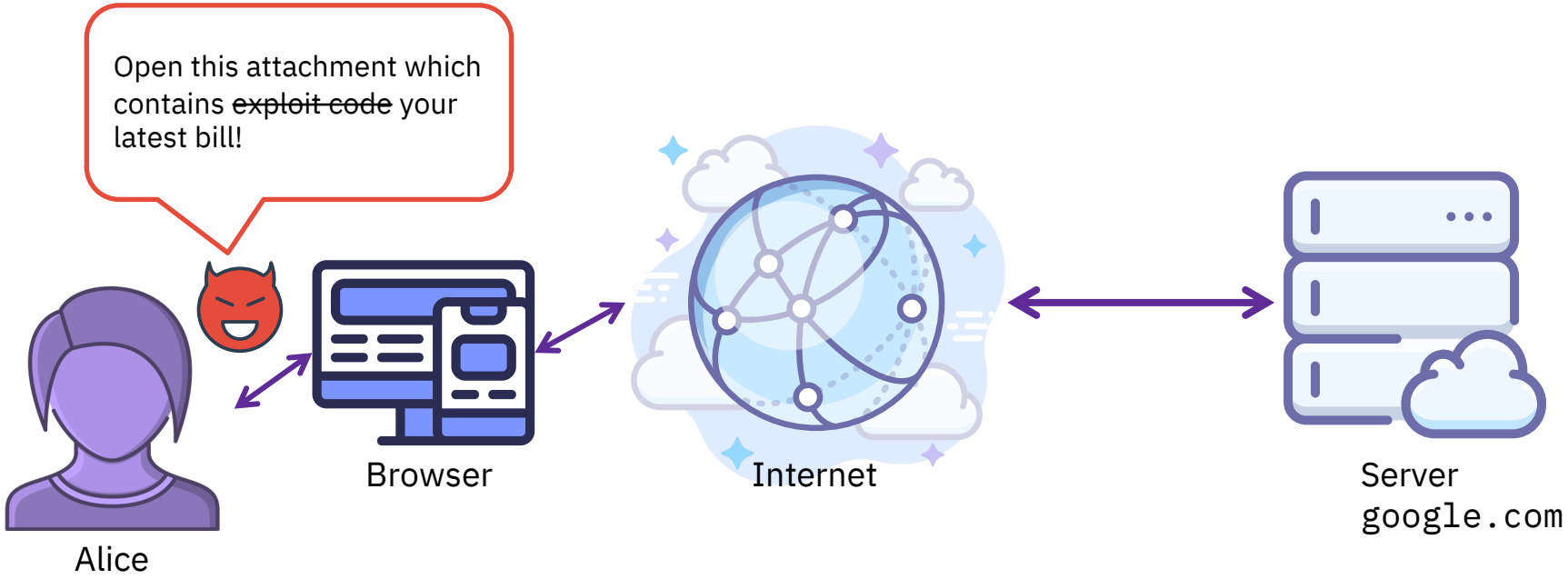
Browser security threats: network attacker.



Browser security threats: software attacker.



Browser security threats: software attacker.



Comparing different attackers.

Web attacker.

- Controls their own website, for which they can get SSL certificates, etc.
- User is misled into visiting the attacker's website. Fundamentally an authentication issue.

Network attacker.

- Passive eavesdropping or active traffic modification. Offset by TLS.

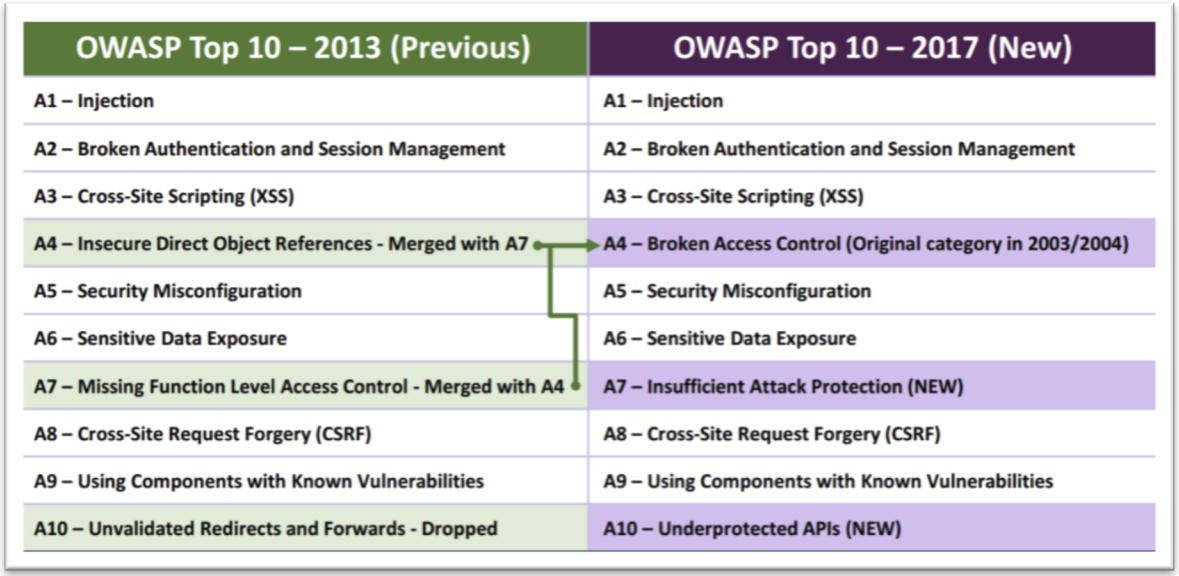
Software attacker.

- Actually finds bugs, tries to run code via XSS, maybe pop a shell...

Comparing different attackers.

Software attacker.

- Actually finds bugs, tries to run code via XSS, maybe pop a shell…
- *Cross-site scripting (XSS)*: inject code into a page that is later executed by a separate client the attacker does not control. (Can you come up with XSS scenarios?)



OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

Browser security goals.

- *Confidentiality*: information on your device is not put at risk simply by browsing the web.
- *Integrity:* Different websites are managed through different sessions in isolation.

Browser Security Mechanisms

4.1b

Browser security mechanisms to cover.

- HTTP.
- Rendering Content.
- Isolation.
- Communication.
- Security User Interface.
- Cookies.

Many more mechanisms exist.

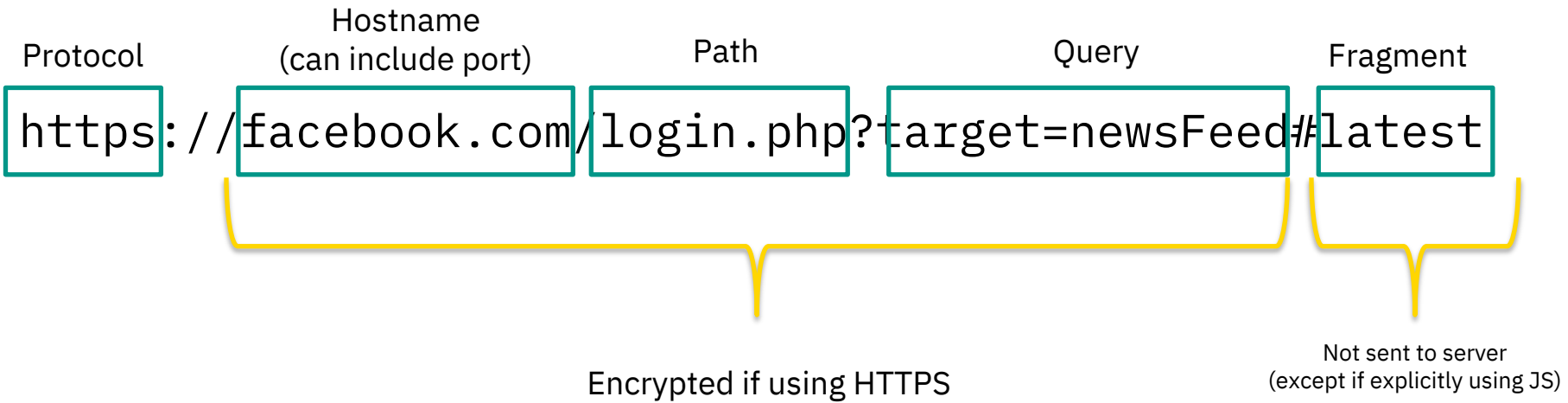
It's impossible to cover them all.

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

URIs.

https://facebook.com/login.php?target=newsFeed#latest

URIs.


Protocol
Hostname
(can include port)
Path
Query
Fragment
https://facebook.com/login.php?target=newsFeed#latest
Encrypted if using HTTPS
Not sent to server
(except if explicitly using JS)

HTTP.

```
GET /index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible;
MSIE5.01; Windows NT)
Host: www.nyu.edu
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

```
POST /login HTTP/1.1
User-Agent: Mozilla/4.0 (compatible;
MSIE5.01; Windows NT)
Host: www.nyu.edu
Content-Type: application/x-www-form-
urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

username=bob&password=logmein23
```

HTTP.

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2018 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2018 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

Examples of HTTP headers related to security.

- *Hypertext Strict Transport Security (HSTS)*: Instructs the browser to only accept HTTPS connections from this domain for the next specified period of time.
- *Content Security Policy (CSP):* Disable dangerous JavaScript and CSS features, prevent loading content from unspecified resource addresses.
- *X-Frame-Options:* Prevent this page from being loaded in an iframe on other websites.

```
Strict-Transport-Security: max-age=<expirationtime>; includeSubDomains

Content-Security-Policy: <directive>

X-Frame-Options: DENY
```

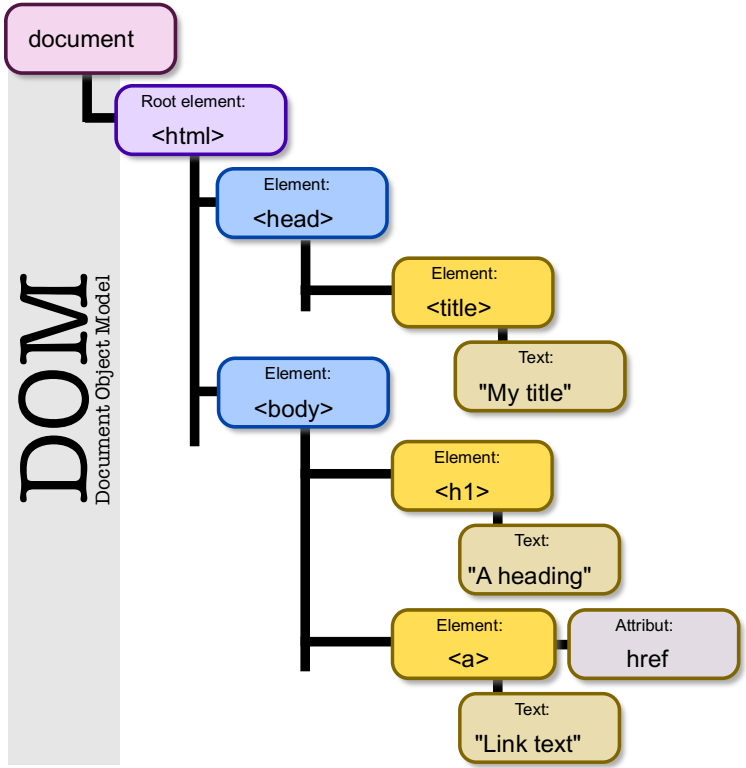
HTTPS and mixed content.

- Even if an entire website is loaded using HTTPS, a single resource being served over HTTP can give the attacker leverage.
- Especially if it's executed.

```
<html>
<head>
<title>NYU</title>
<script type="application/javascript"
src=https://resources.nyu.edu/login.js></scri
pt>
<script type="application/javascript"
src=http://jquerycdn.com/jquery.js></script>
</head>
<body>
</body>
</html>
```

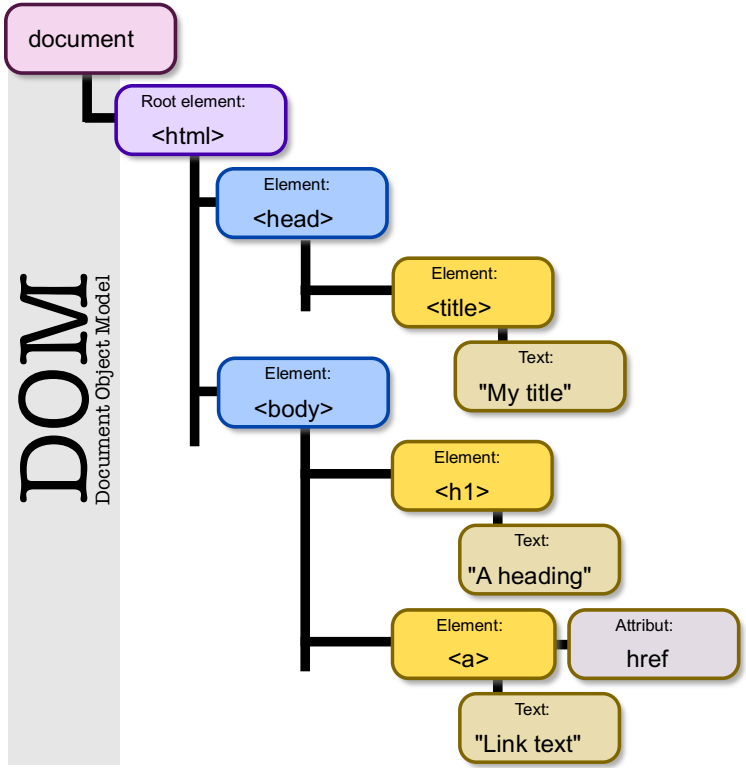
Rendering content.

- Browsers will load content (HTML, XML, CSS) and subsequently render it into the *Document Object Model (DOM).*
- Elements within the DOM can contain content, can have properties and can even trigger events handled by JavaScript code.
- JavaScript is, of course, not rendered but executed.



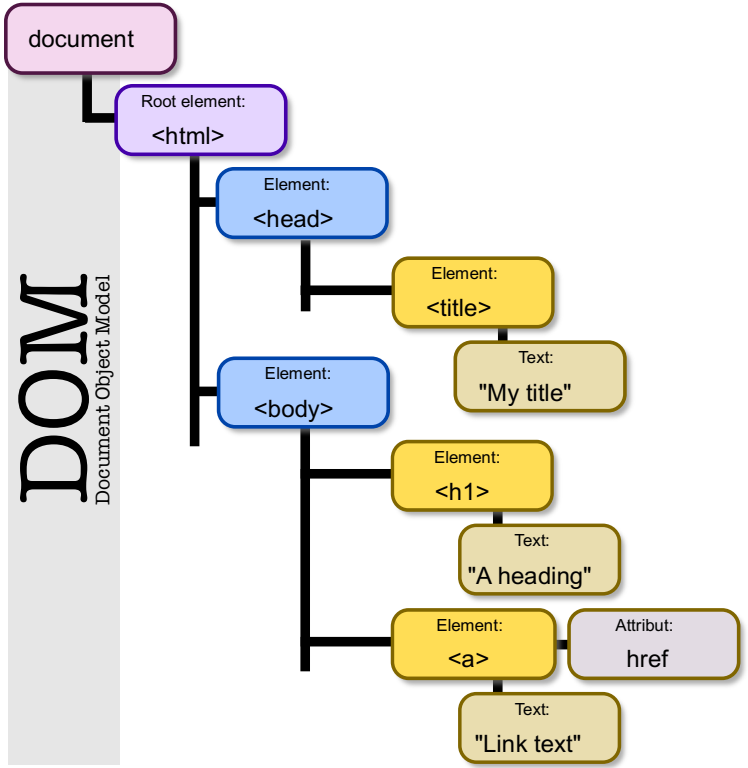
Rendering content.

- Elements in the DOM have methods (like in OO programming): `document.write()`, etc.
- In many ways, the DOM is adjacent to the *Browser Object Model (BOM):* `window, document, history, navigation`…



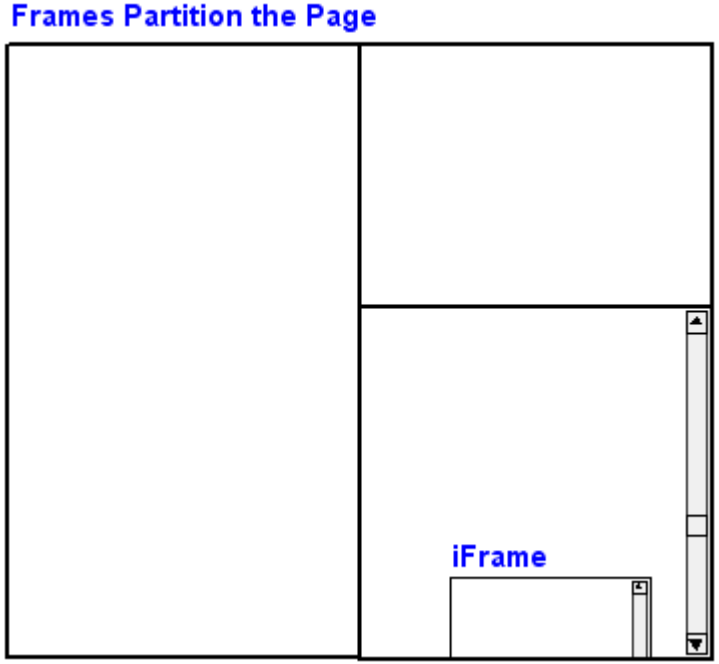
Using JavaScript to learn local information.

- *Example:* Request images from internal IP addresses
- `<img src="192.168.0.4:8080"/>`
- Use timeout/`onError` to determine success/failure.
- Create a map/fingerprint of local systems.



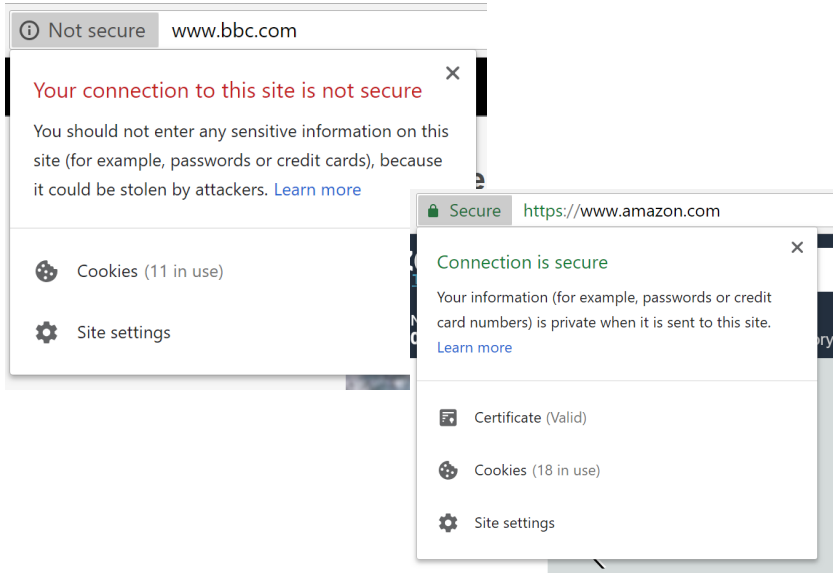
Isolation and Communication.

- *Iframes:* Split a page into separate isolated segments, each with their own namespace.
- Windows and their frames may interact through a restricted API: `window.postMessage`



Security User Interface.

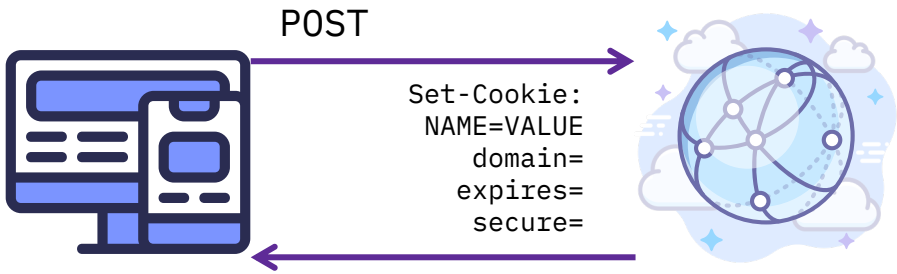
- Users need to check for many markers:
- Is the site using TLS?
- Is the URL accurate?
- Extended validation sometimes helps.
- Users can still be misled by a padlock.jpg.



Cookies.

Cookies act as session identifiers or key-value stores between the web client and web server.

- Once the client logs in, the server may issue them a secret *session cookie* that they both then keep track of.
- *Secure cookies* are sent only over HTTPS.
- *httpOnly cookies* can be sent over HTTP or HTTPS (misleading name) but cannot be accessed by JavaScript via `document.cookies`.



Next time: Web Application Security

Review this learning tool for next time!

https://unescape-room.jobertabma.nl

4.2