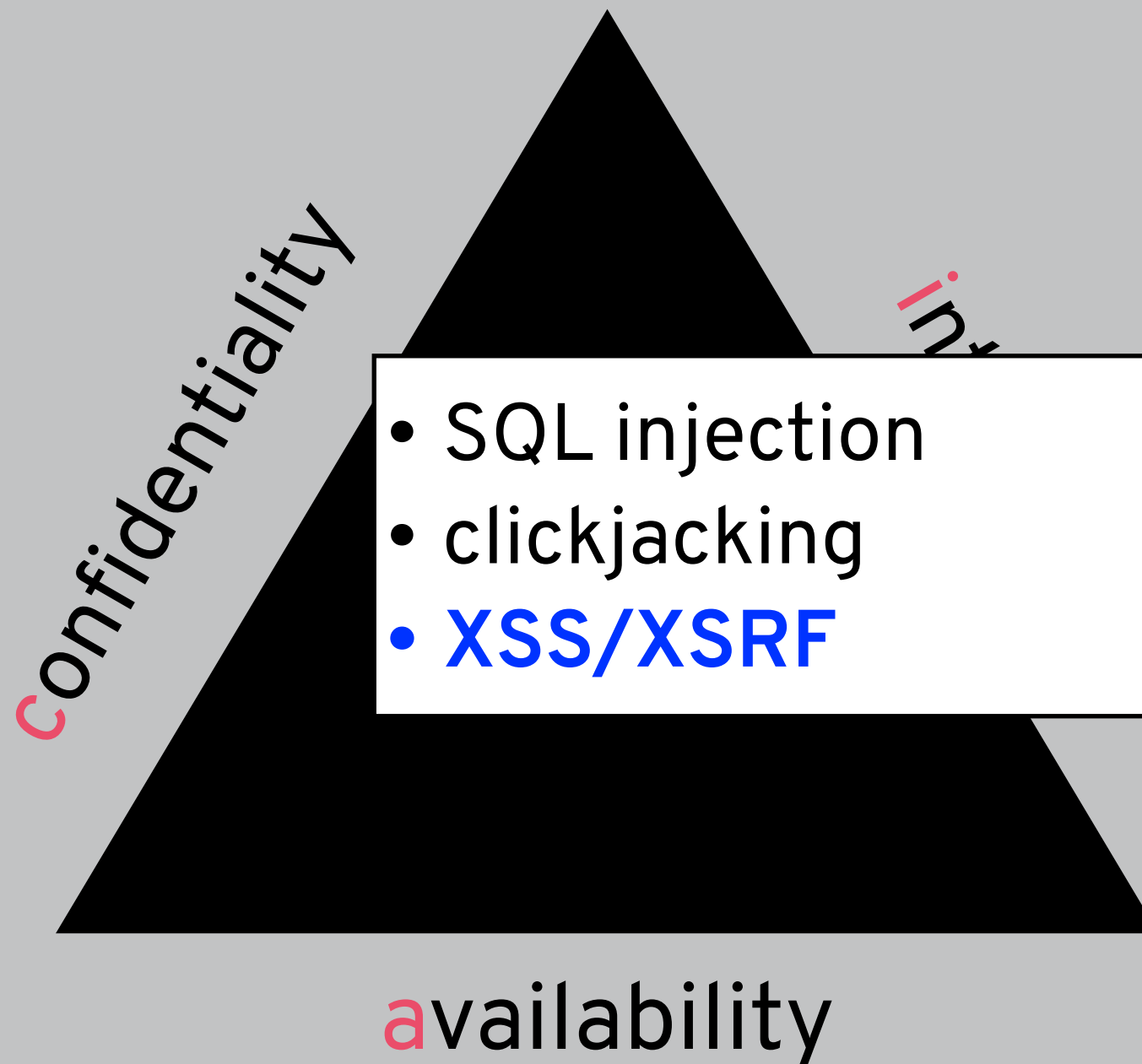

SECURITY (COMP0141):

XSS / XSRF



INTEGRITY, REVISITED



WEB SESSIONS

HTTP is a stateless protocol...

...but most web applications are session-based (you stay logged in until you log out or enough time passes)

How? **Cookies**

WEB COOKIES



The web server provides a token in its response that looks like
`Set-Cookie: <cookie-name>=<cookie-value>`

This is then attached to every future request sent to the server

Examples:

- UserID
- SessionID
- isAuthenticated
- Preferences
- Shopping cart contents
- Shopping cart prices

WEB COOKIES



Session cookies exist only during current browser session

- Deleted when browser is shut down (unless you configure it differently)
- Expiration property is not set

Persistent cookies are saved until some server-defined expiration

What's the **threat model**?

- Who is trusted and who is a potential attacker?
- Does the web browser have to provide cookies?
- How hard is it for a user to modify their cookies?

HOW DOES THE MODERN WEB WORK?


The screenshot shows the top of The New York Times website. The masthead features the newspaper's name in a large, black, serif font. Below it, a navigation bar includes links for World, U.S., Politics, N.Y., Business, Opinion, Tech, Science, Health, Travel, Magazine, T Magazine, and ALL. A large, dark brown banner for car rentals is positioned below the navigation bar. The banner contains a car icon, the text "\$8.95/Day Car Rentals", and a "Book now" button. Three red arrows point from the text "these come from a different site" at the bottom to the car rental banner, the navigation bar, and the weather/stock information. The text "these might too" is placed near the navigation bar. The page is annotated with red boxes and arrows to highlight specific elements.

The School of The New York Times

The New York Times

Tuesday, February 27, 2018 | Today's Paper | Video | 40°F | FTSE 100 +0.02% ↑

World U.S. Politics N.Y. Business Opinion Tech Science Health **these might too** Travel Magazine T Magazine ALL

 **\$8.95/Day Car Rentals** [Book now](#)

\$8.95/Day Car Rentals Compare & Book Deals Up to 35% Off!

these come from a different site

JAVASCRIPT

JavaScript was designed as scripting language for Navigator 2, implemented in (literally) 10 days and related to Java in name only (“Java is to JavaScript like car is to carpet”)

Scripts embedded in web pages using `<script>` tag that get the browser to execute some linked script (`src="function.js"`)

This means your computer is executing code (scripts) that it finds on the Internet

JAVASCRIPT SECURITY

Script runs in a “sandbox”: no direct file access and restricted network access

Same-origin policy: script can read properties of documents only from the same server, protocol, and port

But, same-origin policy **does not apply** to scripts loaded from arbitrary site, so `<script type="text/javascript" src="http://www.sarah.com/myscript.js"></script>` runs as if it were loaded from the site that provided the page!

Server can also explicitly tell browser other domains that are allowed using `Access-Control-Allow-Origin` header

HTML INJECTION

Many interactive web applications echo user input

- Search queries
- Tweets
- Forum posts



asdfghjkl



All

Images

Videos

News

Maps

More

Settings

Tools

About 7,790,000 results (0.30 seconds)



A privacy reminder from Google

REMIND ME LATER

REVIEW

Urban Dictionary: asdfghjkl

<https://www.urbandictionary.com/define.php?term=asdfghjkl>

An expression used when you are so excited you can't even find the words to describe your feelings.

People also ask

What does Asdfghjkl mean?



What does Zxcvbnm mean?



HTML INJECTION

Many interactive web applications echo user input

- Search queries
- Tweets
- Forum posts

What if user input contains HTML markup tags?

Similar story as with SQL injection: if the server doesn't sanitise and encode it, markup is rendered by the web browser as it is provided by any user of the website

ALERT(1) TO WIN

alert(1) to win

The code below generates HTML in an unsafe way. Prove it by calling `alert(1)`.

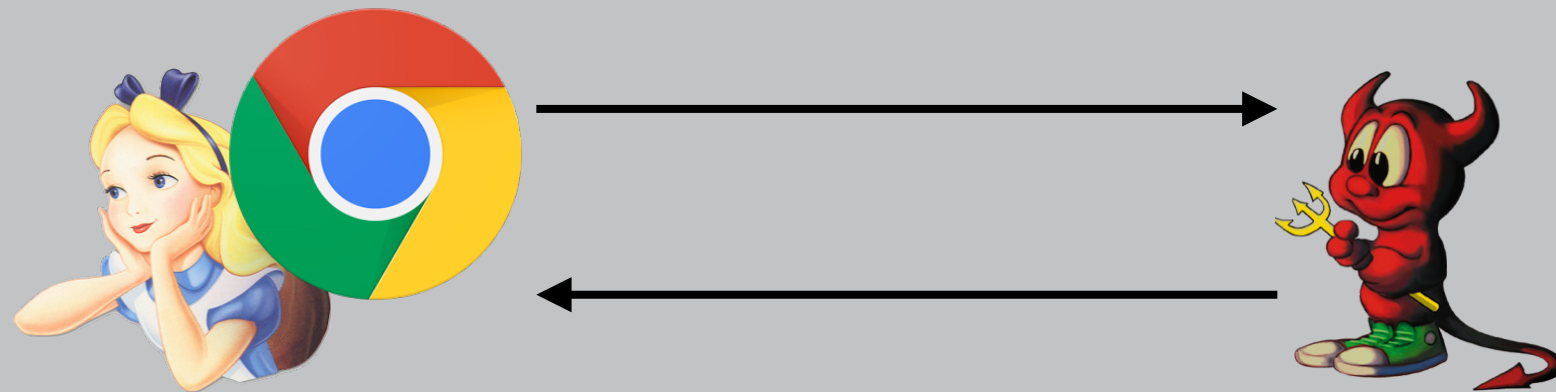
```
function escape(s) {  
  return '<script>console.log("'" + s + "'");</script>';  
}
```

Input

type here



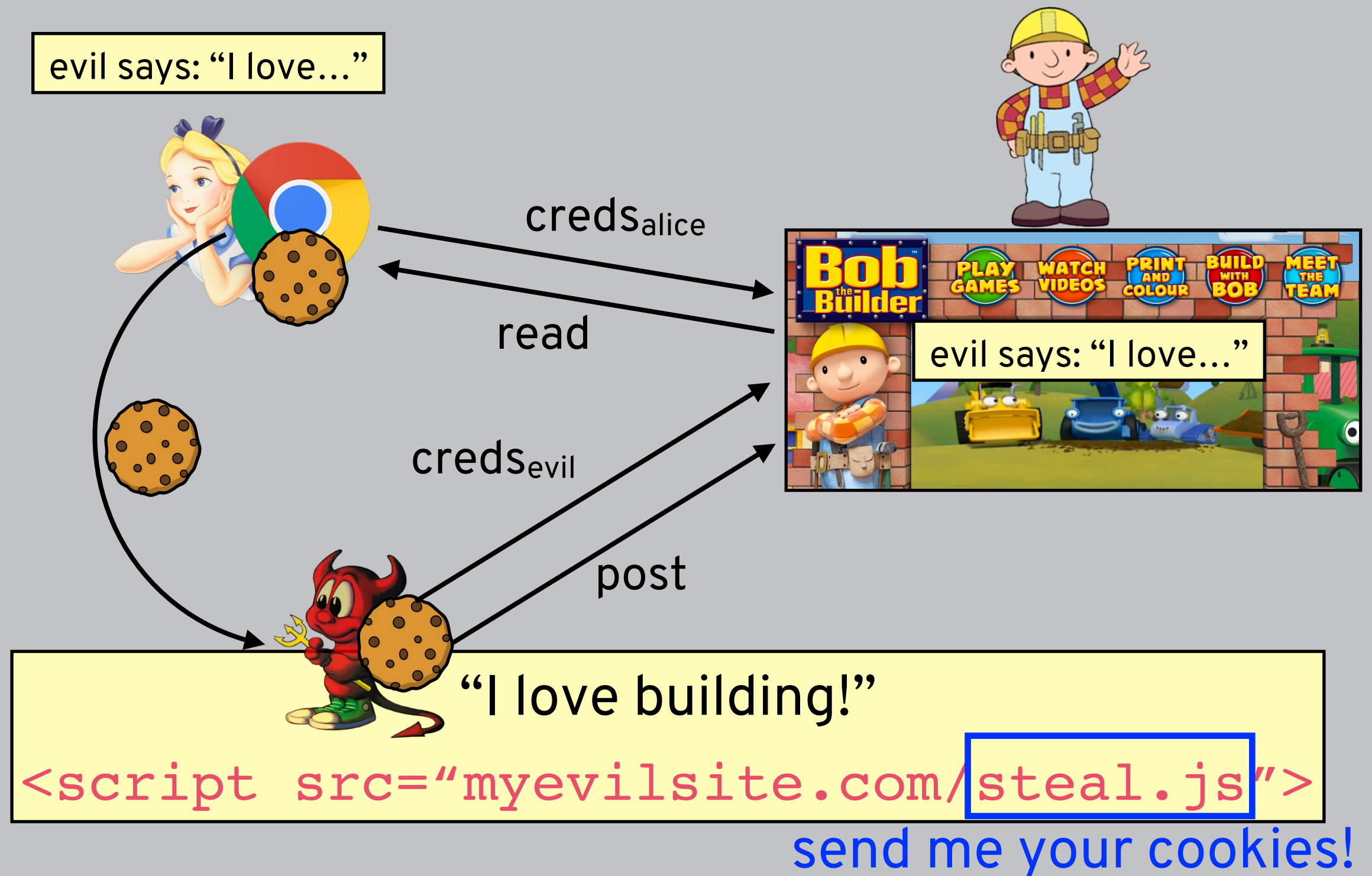
THREAT MODEL



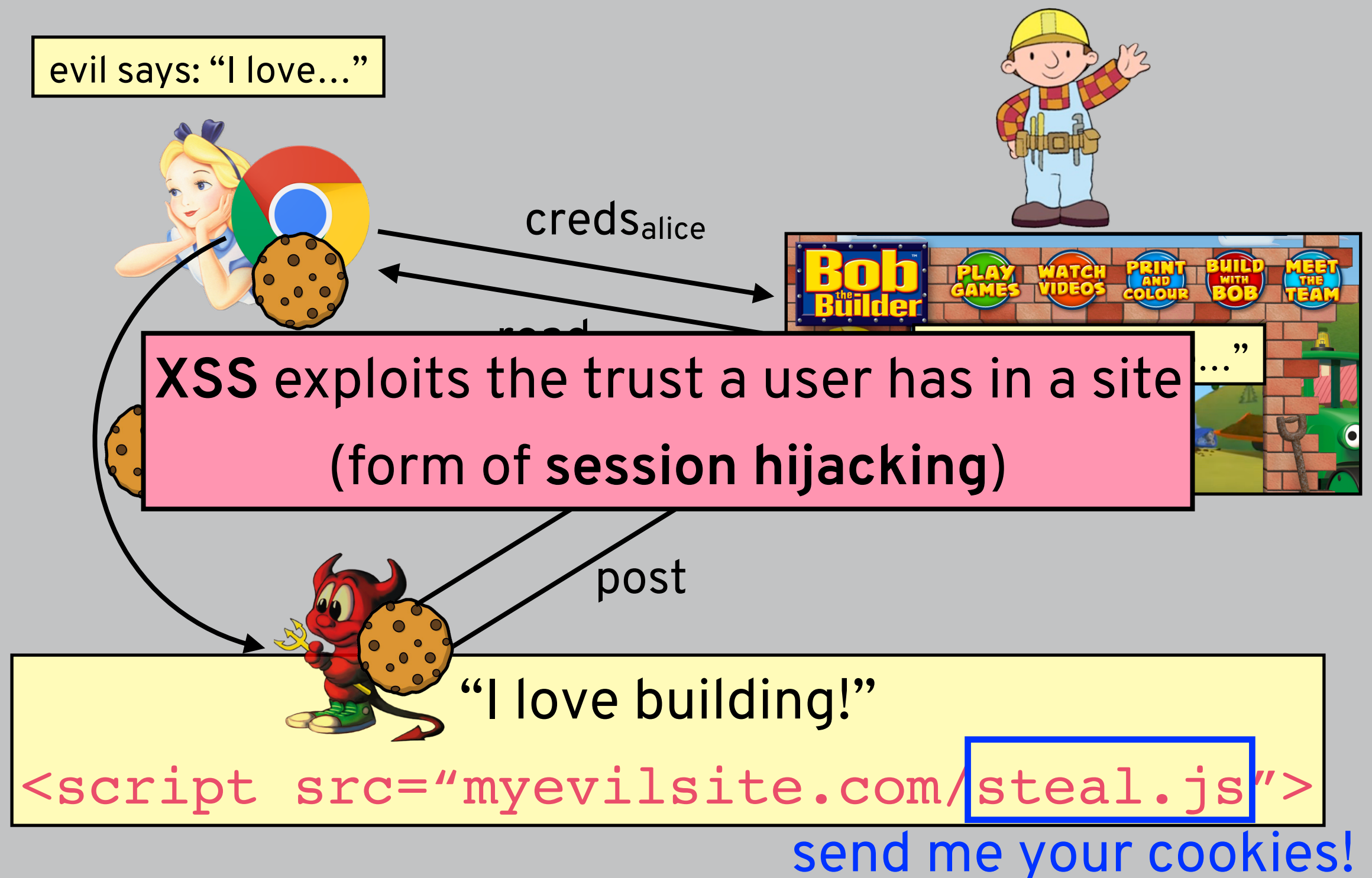
Is the server trusted by the browser? or the user?

- Browser fingerprinting
- Forward secrecy / revocation
- Typosquatting / pharming
- Clickjacking
- XSS (trusted to be careful, not just non-malicious!)

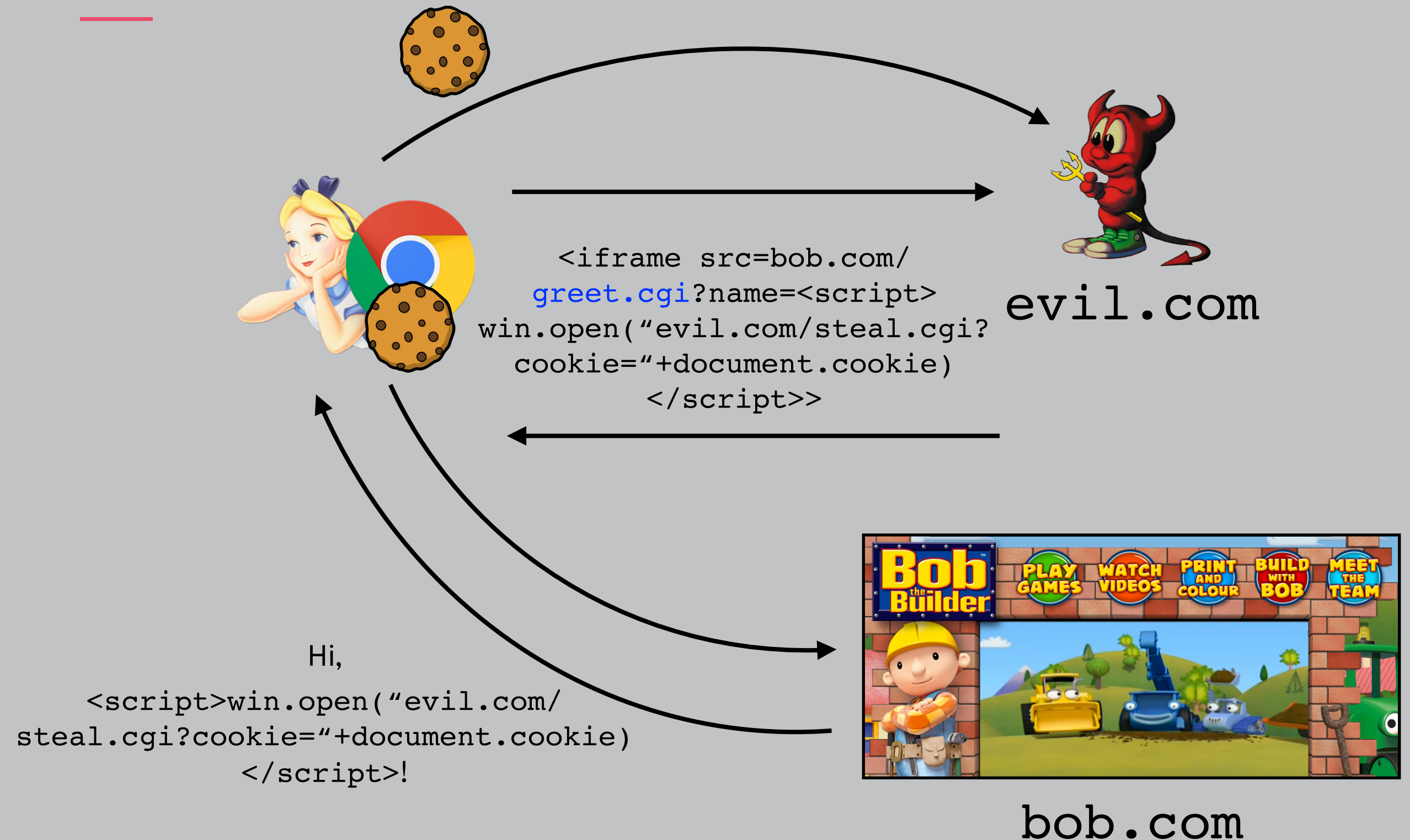
CROSS-SITE SCRIPTING (XSS)



CROSS-SITE SCRIPTING (XSS)



REFLECTED XSS



CROSS-SITE SCRIPTING (XSS)

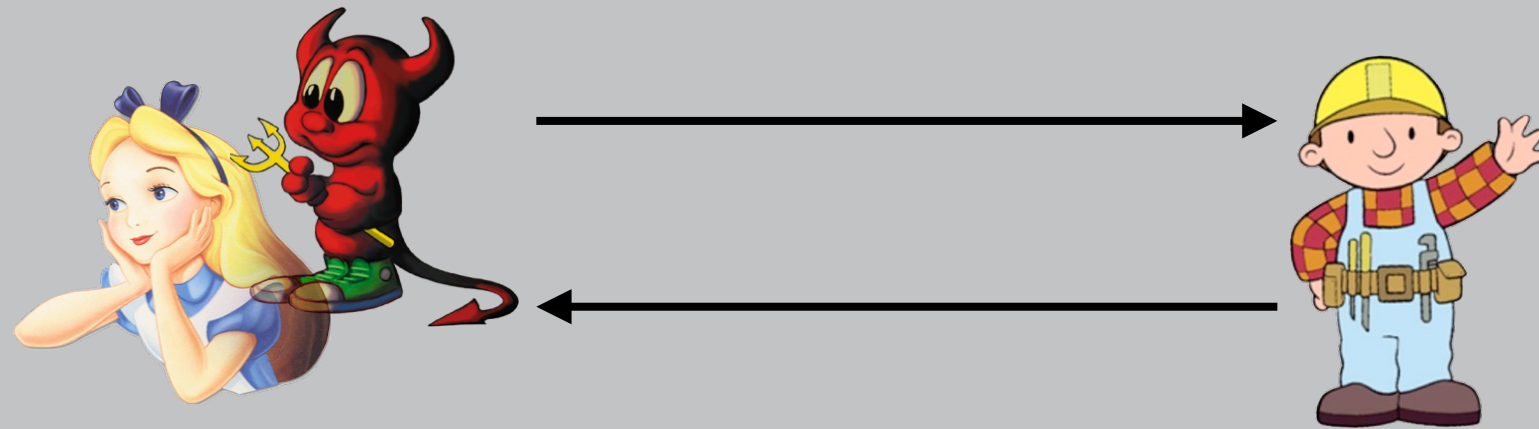
More generally, XSS lets an attacker do anything a legitimate client-side script from that server could do

- Show false information
- Request sensitive information
- Trigger HTTP requests from the client

How to prevent?

- Preventing injection of scripts is hard! Not enough to block “<” and “>” or allow only simple HTML tags
- Partial fix: `httpOnly` cookies cannot be accessed via script (but this doesn't stop XSS attacks, just cookie theft)

THREAT MODEL

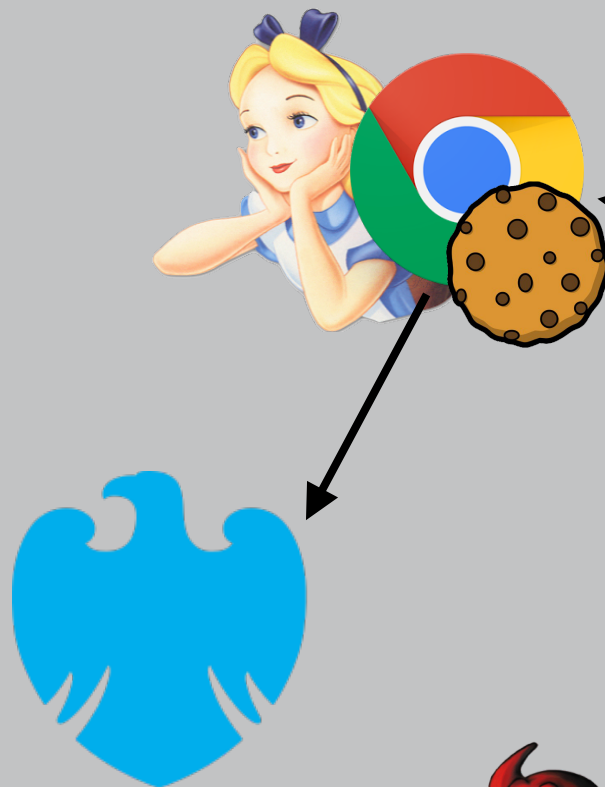


Is the browser trusted by the user? or the server?

- XSRF (trusted to be careful, not just non-malicious!)

CROSS-SITE REQUEST FORGERY (XSRF)

evil says: "I love..."



read



post



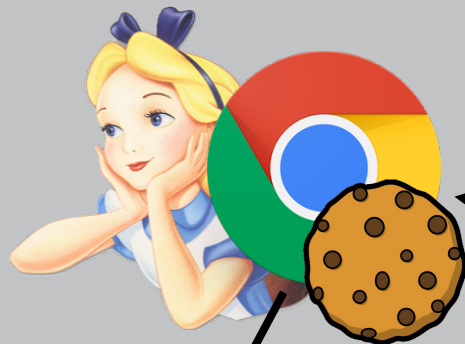
"I love building!"

```

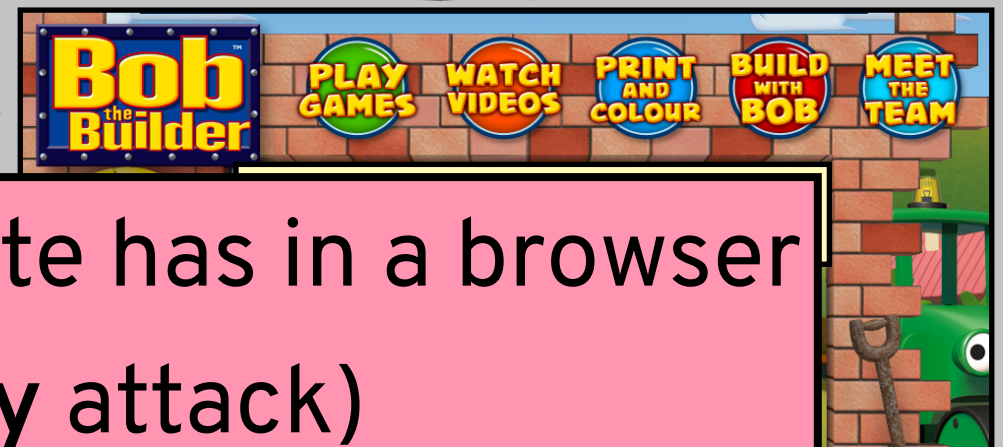
```

CROSS-SITE REQUEST FORGERY (XSRF)

evil says: "I love..."



read



**XSRF exploits the trust a site has in a browser
(confused deputy attack)**

post



"I love building!"

```

```

CROSS-SITE REQUEST FORGERY (XSRF)

When a browser issues a GET request, it attaches all cookies it has from the target site

The target sees the cookies but has no way of knowing the request was really authorised by the (human) user

How to prevent?

- Secret tokens visible only by same-origin content (client needs to include these tokens in state-altering requests)
- Don't alter state based on GET requests
- Same-origin cookies (Chrome)

XSS VS. XSRF

XSS

- Server-side vulnerability
- Attacker injects a script into the trusted website
- Trusting browser executes attacker's script

XSRF

- Server-side vulnerability
- Attacker gets trusted browser to issue requests
- Trusting website executes attacker's requests

MITIGATIONS

for websites:



use same-origin policy / content security policy

sanitise HTML

require additional authentication

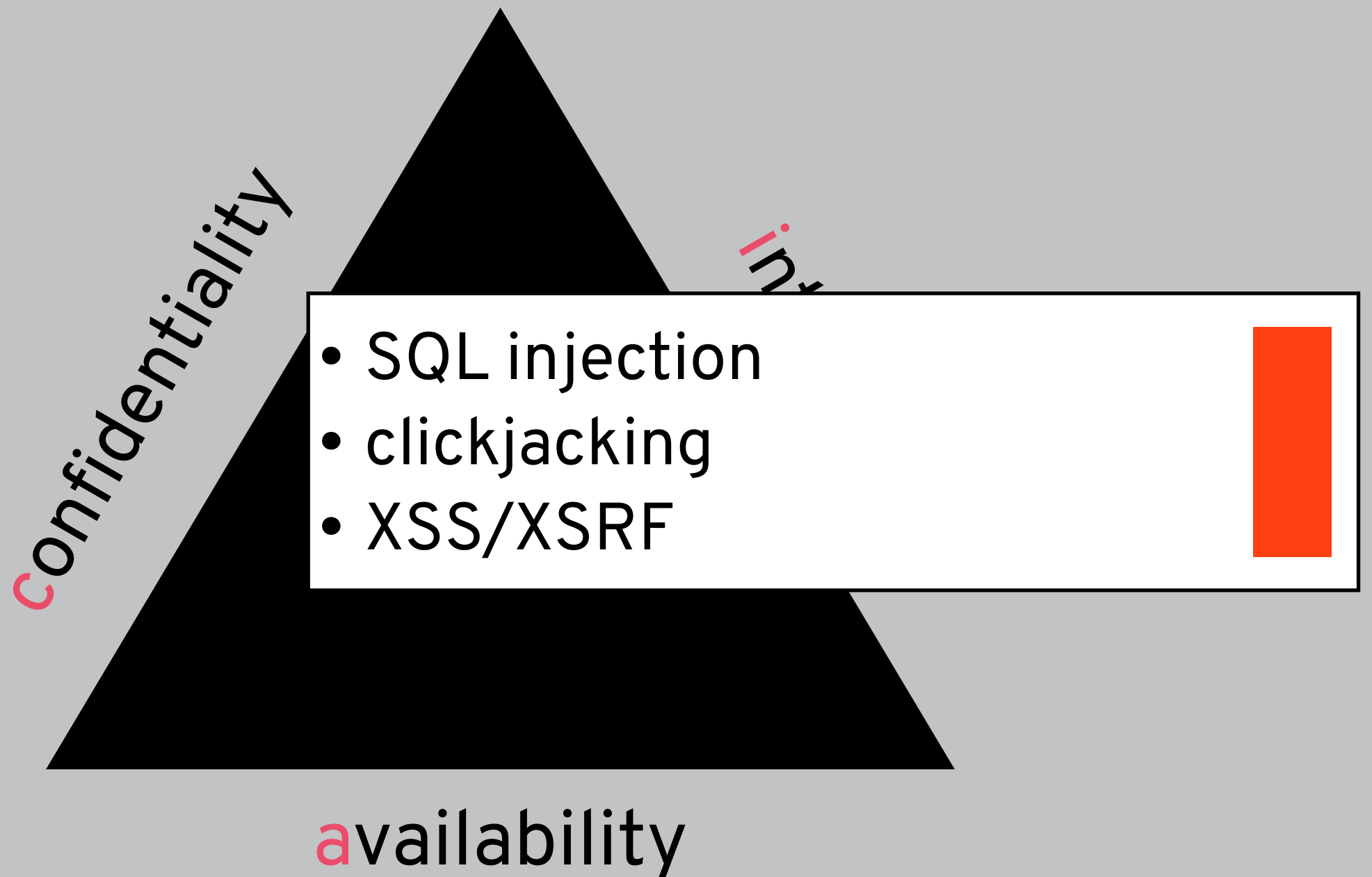
for users:



don't run scripts! (NoScript, Ghostery, etc.)

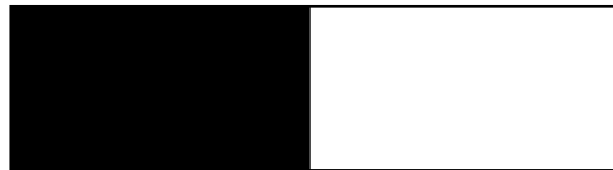
don't "stay signed in"

INTEGRITY, REVISITED



CIA TRIANGLE

defences use cryptography
and are (largely) comprehensive



confidentiality

integrity

defences use system attributes
and are more ad hoc



availability

QUIZ!

Please go to

`https://moodle.ucl.ac.uk/mod/quiz/view.php?id=2885316`

to take this week's quiz!