Imperial College
London

Network and Web Security

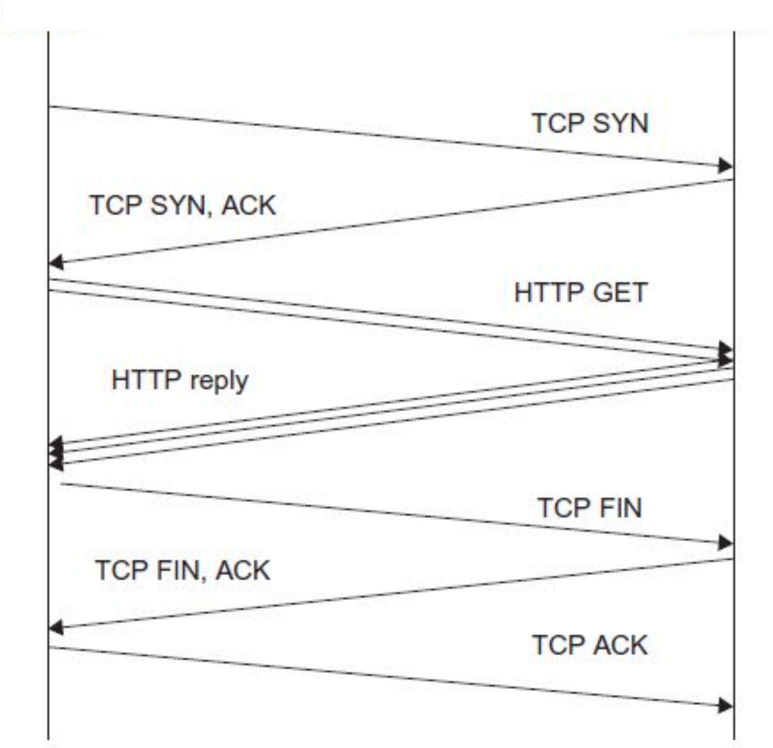
Browser storage

Dr Sergio Maffeis
Department of Computing
Course web page: https://331.websec.fun

HTTP state management

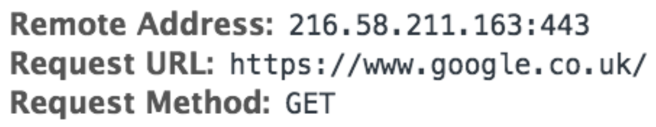
Imperial College London

- HTTP is a stateless protocol
  - Each request-response exchange is independent of the previous one
  - Compare with TCP: after initial handshake, data-ack packets are tied by increasing session numbers
- Web applications need to keep their own state
  - Server-side: can save files, access databases
  - Client-side: in 1994 Netscape introduced Cookies
    - Key-value pairs stored by the browser on behalf of a web application
- Cookies are used for
  - Storing website preferences
  - Storing session tokens
  - Tracking users
- Current specification is IETF RFC 6265, from 2011
  - Some basic guarantees on browser implementations
    - “At least 4096 bytes per cookie”
    - “At least 50 cookies per domain”
    - “At least 3000 cookies total”



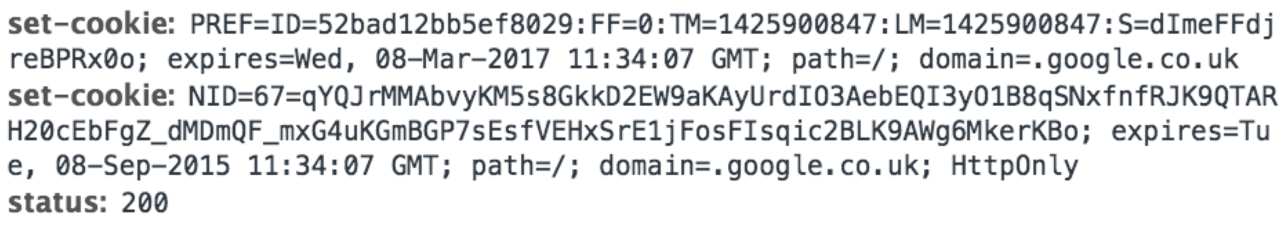
Cookies in HTTP

- First client request carries no cookies



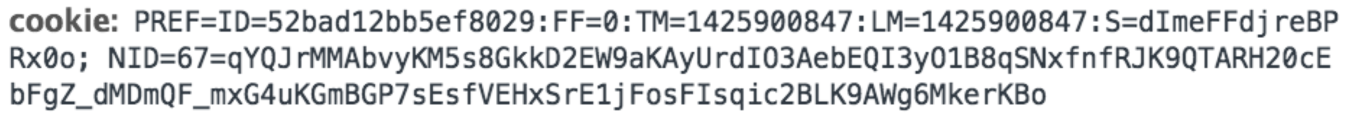
```
Remote Address: 216.58.211.163:443
Request URL: https://www.google.co.uk/
Request Method: GET
```

- The server sets each cookie using a response header
  - `Set-Cookie: name = value; [(attribute [= value];)*]`
  - Optional attributes tell browser how to handle that cookie



```
set-cookie: PREF=ID=52bad12bb5ef8029:FF=0:TM=1425900847:LM=1425900847:S=dImeFFdj
reBPRx0o; expires=Wed, 08-Mar-2017 11:34:07 GMT; path=/; domain=.google.co.uk
set-cookie: NID=67=qYQJrMMAbvyKM5s8GkkD2EW9aKAyUrdIO3AebEQI3yO1B8qSNxfnfRJK9QTAR
H20cEbFgZ_dMDmQF_mxG4uKGmBGP7sEsfVEHxSrE1jFosFIsqic2BLK9AWg6MkerKBo; expires=Tu
e, 08-Sep-2015 11:34:07 GMT; path=/; domain=.google.co.uk; HttpOnly
status: 200
```

- Browser includes relevant cookies for subsequent requests in one request header
  - `Cookie: (name = value;)`$^+$ …



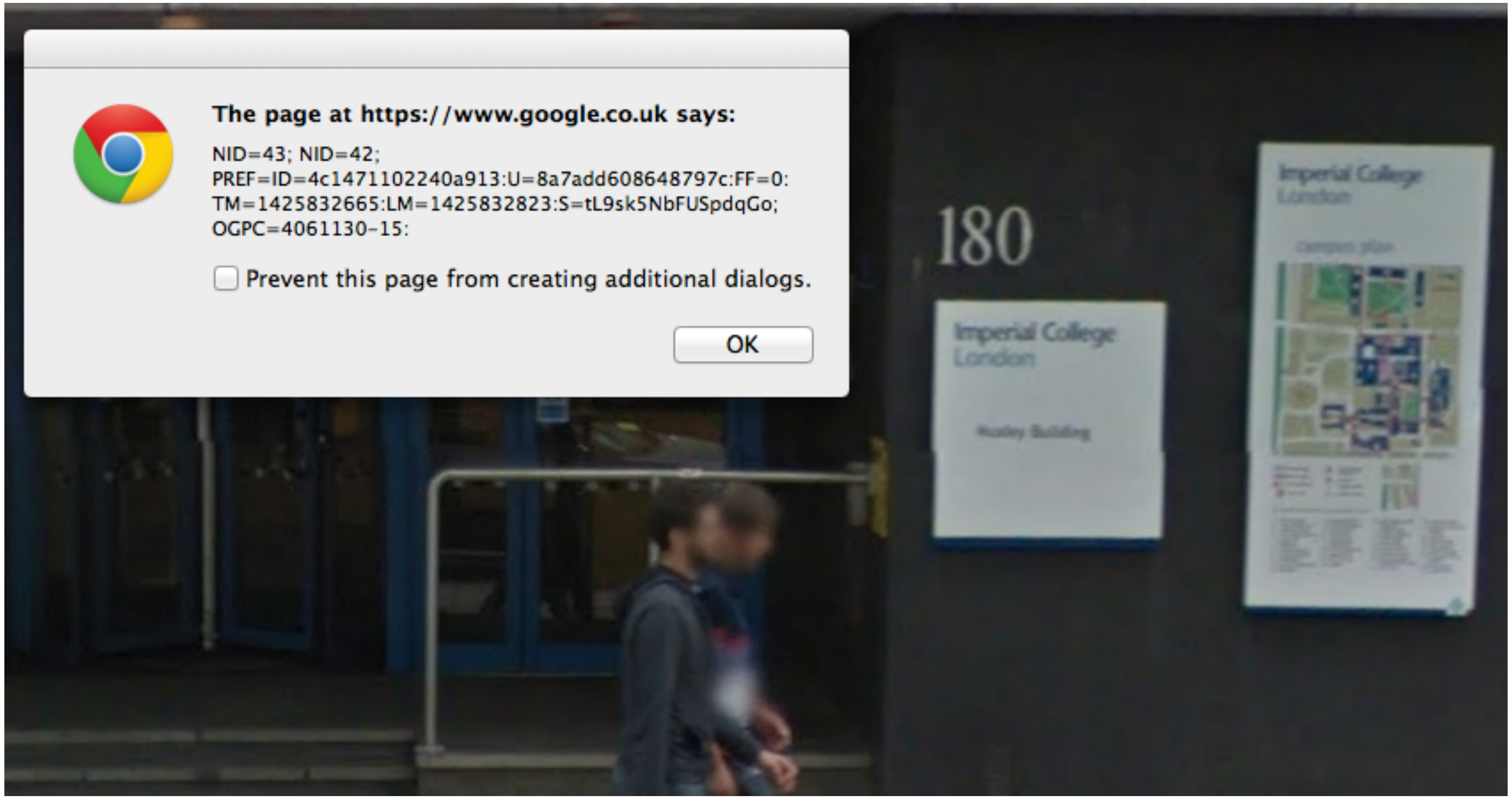
```
cookie: PREF=ID=52bad12bb5ef8029:FF=0:TM=1425900847:LM=1425900847:S=dImeFFdjreBP
Rx0o; NID=67=qYQJrMMAbvyKM5s8GkkD2EW9aKAyUrdIO3AebEQI3yO1B8qSNxfnfRJK9QTARH20cE
bFgZ_dMDmQF_mxG4uKGmBGP7sEsfVEHxSrE1jFosFIsqic2BLK9AWg6MkerKBo
```

Cookie attributes

- `Domain = `*`domain`*
  - Send cookie back when *`domain`* is suffix of requested domain
    - Cookie for `example.com` is sent to `login.example.com`
  - Must be a non-trivial suffix of host domain: cannot set for "`.com, .co.uk, .googlecode.com`"
    - The Public Suffix List is used to keep track of valid suffixes
  - Defaults to host of URL that caused the response
- `Path = `*`path`*
  - Send cookie back when *`path`* is prefix of request path
    - Cookie for `example.com/login` is not sent to `example.com/`
  - Defaults to path of URL that caused the response
- `Expires = `*`date`* (similar for `Max-Age`)
  - Date in the future: store *persistent* cookie on file until *`date`*
  - Date in the past: delete the cookie immediately
  - `Null` (default): keep *session-only* cookie in memory until browser is closed
- `Secure`
  - Send the cookie only over HTTPS
  - Provides confidentiality against network attacker
- `HttpOnly`
  - Prevent non-HTTP APIs (for example JavaScript) from accessing cookie
  - Mitigates risk of cookie theft via XSS
- SameSite
  - Experimental feature, already supported by Chrome, Firefox and Opera
  - Mitigates cross-origin information leakage (for example CSRF and tracking, as we shall see later in the course)
  - `Strict`: cookie is sent only from a page with same domain
  - `Lax`: don't block for top-level cross-domain navigation with safe HTTP methods (GET, OPTIONS, HEAD, TRACE)
  - `None`: as of February 2020 Chrome assumes `Lax` by default, unless one specifies `None`, and uses also attribute `Secure`

Cookies in the browser

- Cookies are also accessible to JavaScript in the browser
  - `document.cookie` provides access to all the cookies **in scope** for the document origin
- Examples
  - Write: `document.cookie = "userid=123; path=/; secure";`
  - Delete: `document.cookie = "userid=; path=/; expires=Thu, 01 Jan 1970 00:00:01 GMT”;`
  - Read: `alert(document.cookie);`   (this will not show `httponly` cookies)



The page at https://www.google.co.uk says:

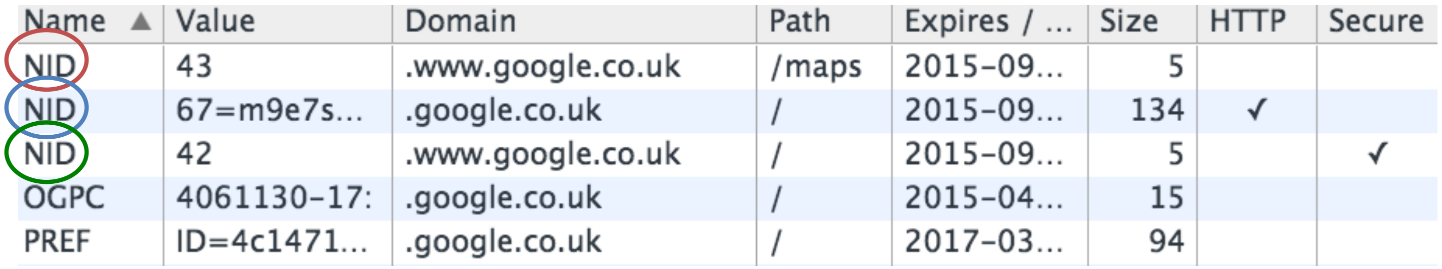
```
NID=43; NID=42;
PREF=ID=4c1471102240a913:U=8a7add608648797c:FF=0:
TM=1425832665:LM=1425832823:S=tL9sk5NbFUSpdqGo;
OGPC=4061130-15:
```

□ Prevent this page from creating additional dialogs.

Cookie scope

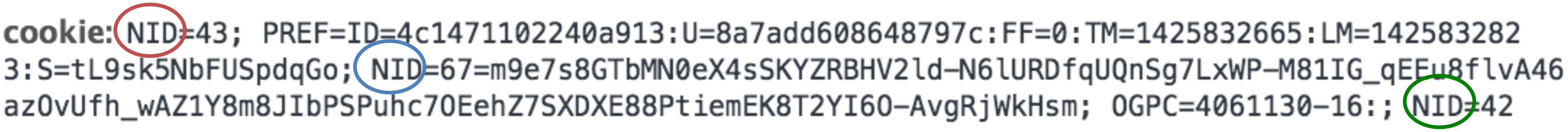
- Cookie origin: domain, path
- Cookie is identified by name and origin
- Cookie scope is determined by origin and secure attribute
- Browser request sends all cookies that are in scope to server
  - All the server sees is the name=value pairs
  - Attributes are not sent back
- Example
  - We added 2 “NID” cookies (42,43) for different Google origins using JavaScript

Cookie store for https://www.google.co.uk/maps



Name ▲	Value	Domain	Path	Expires / …	Size	HTTP	Secure
NID	43	.www.google.co.uk	/maps	2015-09…	5		
NID	67=m9e7s…	.google.co.uk	/	2015-09…	134	✓	
NID	42	.www.google.co.uk	/	2015-09…	5		✓
OGPC	4061130-17:	.google.co.uk	/	2015-04…	15		
PREF	ID=4c1471…	.google.co.uk	/	2017-03…	94		

Request header for https://www.google.co.uk/maps



```
cookie: NID=43; PREF=ID=4c1471102240a913:U=8a7add608648797c:FF=0:TM=1425832665:LM=142583282
3:S=tL9sk5NbFUSpdqGo; NID=67=m9e7s8GTbMN0eX4sSKYZRBHV2ld-N6lURDfqUQnSg7LxWP-M81IG_qEEu8flvA46
azOvUfh_wAZ1Y8m8JIbPSPuhc7OEehZ7SXDXE88PtiemEK8T2YI6O-AvgRjWkHsm; OGPC=4061130-16:; NID=42
```

Security considerations

- Server does not see cookie attributes
  - `example.com` cannot tell if cookie was set by `subdomain.example.com`
  - Cannot tell if cookie is effectively `httponly` or was written by JavaScript
- Path does not restrict visibility of cookies
  - Path in cookie origin is only meant to improve efficiency
    - Send to server only the cookies that are needed for a specific request
  - Scripts from different path can
    - Load iframe with page from target path
    - Access `document.cookie` of iframe thanks to SOP (which ignores path)
- Cookie integrity is not guaranteed
  - User can access cookies SQLite database in browser
  - Any JavaScript from the same origin can set/edit cookies
  - Even **`secure`** does not guarantee cookie integrity
    - Secure cookies can be set by JavaScript (risk of XSS, injection)
    - Active network attacker can intercept HTTPS response and set cookie with spoofed HTTP response: next request sends tampered “secure” cookie

HTML5 browser storage

- HTML5 also provides Web Storage and Indexed Database APIs.
  - We look at Web Storage (easier to use, larger adoption)
  - Implements client-side state using lists of key-value pairs
- `window.localStorage`
  - Associated to page origin
  - Data is kept until it is deleted explicitly
- `window.sessionStorage`
  - Associated to current tab **and** page origin
  - Data is kept until the tab is closed
- JavaScript API (where `xxx = localStorage` or `sessionStorage`)
  - `xxx.length //return list length`
  - `xxx.key(n) //read n-th key in list`
  - `xxx.getItem(k) or xxx.k //read value of k`
  - `xxx.setItem(k,v) //set value v for key k`
  - `xxx.removeItem(k) //remove entry with key k`
  - `xxx.clear() //delete all entries`
- HTML5 storage is not sent/set over HTTP
  - Up to the page if/how to involve the server (AJAX, GET/POST, etc.)
  - Attacker model is script injection, XSS

Attack: Resident XSS

Target page:

```
<html><body>
Welcome user:
<script>
    document.write(localStorage.getItem("user_name"));
</script>
</body></html>
```

Attack vector:

```
localStorage.setItem("user_name","<script>alert('XSS!')</script>");
```

- Resident XSS (RXSS) is a variant of DOM-based XSS that exploits browser storage
  - Cookie, Web storage, Indexed DB, etc.
- Attacker must already be able to inject JavaScript to exploit the RXSS
  - **RXSS remains effective also after vulnerable page is patched**
    - Unlike DOM-based and Reflected XSS
  - RXSS cannot be detected by server, IDS, XSS Auditor
    - Unlike Reflected, Stored XSS
- Countermeasure: do not trust values stored in the browser (*defense-in-depth*)
  - Sanitise stored values like other user input
  - Periodically validate, refresh or delete stored data