Imperial College
London

CO331 – Network and Web Security

10. HTTP

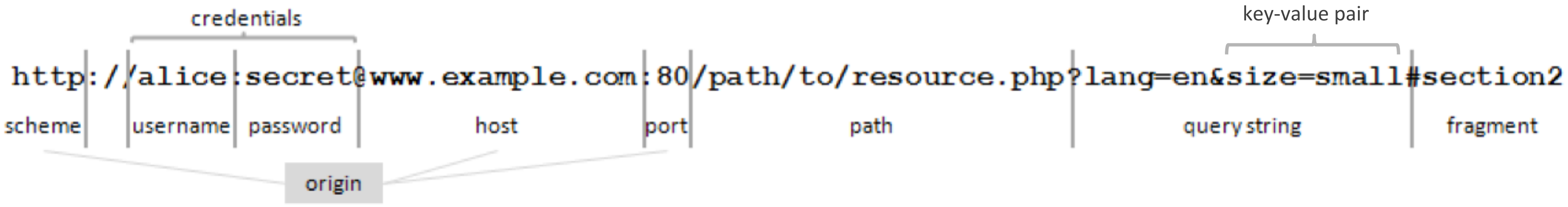
Dr Sergio Maffeis
Department of Computing
Course web page: http://www.doc.ic.ac.uk/~maffeis/331

URLs

`https://host1.example.com:5588/private/login.php`

- Uniform Resource Locators
- *Scheme* specifies what protocol to use
  - Many options: 238 and counting…
    - Main ones: `http, https, ftp, javascript, mailto, chrome, data` …
    - “Full” list at http://www.iana.org/assignments/uri-schemes/
- *Host* is the target IP address
  - Or hostname, that needs to be resolved via DNS
- *Port* identifies the port on the target
  - If unspecified, it defaults to the standard port for the scheme
    - 80 for HTTP, 443 for HTTPS, 21 for FTP
- *Path* denotes the requested resource
  - An image, a HTML file, the output of running a PHP script, …
- **Origin = (scheme, port, host)**: crucial concept for web security!!!

URLs

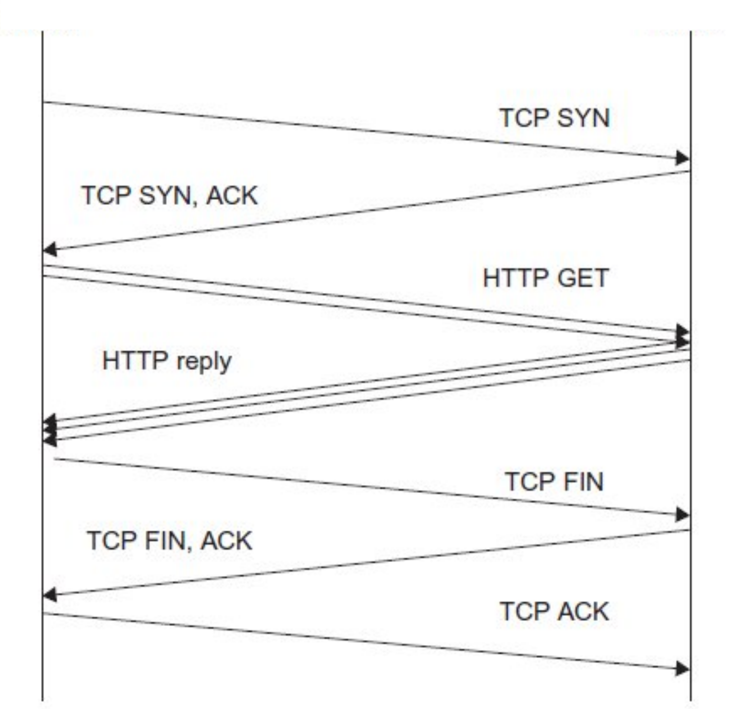

credentials
key-value pair
http://alice:secret@www.example.com:80/path/to/resource.php?lang=en&size=small#section2
scheme
username
password
host
port
path
query string
fragment
origin

- **Credentials** are used in a protocol-dependent way
  - If absent, defaults to anonymous access
- **Query string** contains parameters that are passed to the resource handler
- **Fragment** remains on the client
  - Tells browser to scroll to a specific point in an HTML document
- In practice, it’s up to the client and server how to interpret the fields of a URL
  - Query strings can be anything, so careful about misinterpreting `http://a.com?{b:"a5 = z; "}`
  - We shall see examples when we talk about the browser
- Security considerations
  - URIs contain key information for web applications
    - We care bout the confidentiality of credentials
    - We care about integrity of the path (REST requests have side effects on server)
    - We care about integrity and confidentiality query string (may be sensitive data)
  - Parsing URIs incorrectly may lead to security issues
    - Quiz: what is the **origin** of these requests?
    - http://a.com#b:c@d.com
    - http://a.com:b:c@d.com

HTTP/1.1

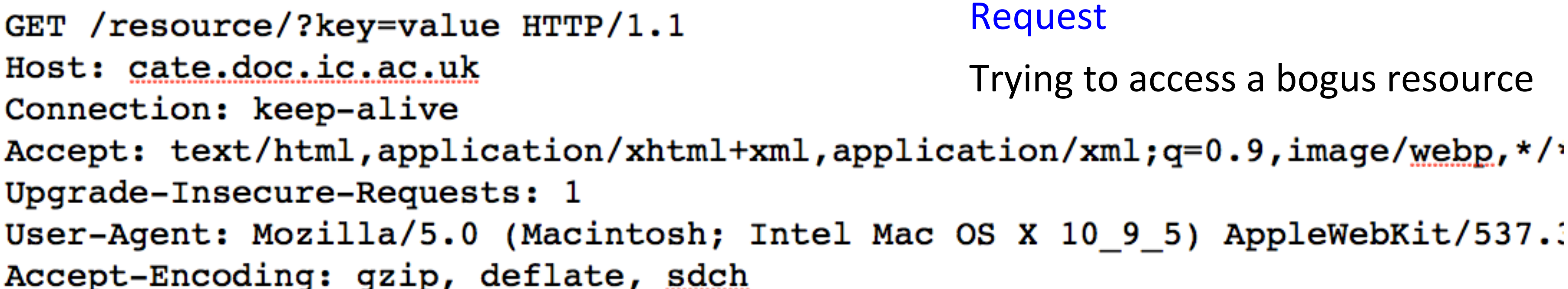
Imperial College London

- Client-server protocol
  - Client initiates a TCP connection
  - Client sends a request conforming to HTTP protocol format
  - Server replies with a protocol-specific response
    - Typically containing data or an error message
  - Server closes the TCP connection
- *Keepalive*: for efficiency, the TCP connection is now kept open for a few seconds, in case there is a follow-up request
- Yet, the protocol is stateless
  - Each request is handled independently of the previous request
  - It's up to client and server to maintain state
    - Cookies help: much about them later…
- Main *methods:* GET, POST
  - Less common: HEAD, PUT, DELETE, CONNECT, TRACE, OPTIONS
  - Possible to add custom methods



GET

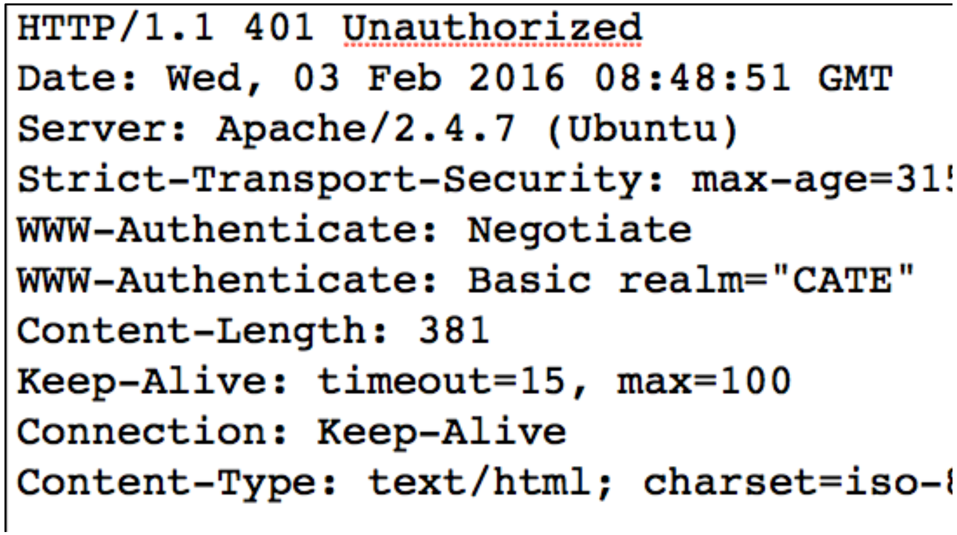
- Fetch a resource from the server
  - Can pass parameters in the query string
  - Empty body
  - Originally meant to be side-effect free and idempotent
    - In practice, it's up to the server to decide



```
GET /resource/?key=value HTTP/1.1
Host: cate.doc.ic.ac.uk
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

Request

Trying to access a bogus resource



```
HTTP/1.1 401 Unauthorized
Date: Wed, 03 Feb 2016 08:48:51 GMT
Server: Apache/2.4.7 (Ubuntu)
Strict-Transport-Security: max-age=31
WWW-Authenticate: Negotiate
WWW-Authenticate: Basic realm="CATE"
Content-Length: 381
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-
```

Response

If I was logged in on CATE, response body would countain an HTML page

POST

- Submit data to the server
  - Contains a body with the payload
  - Can still pass parameters in the query string
    - Standard web forms use the body instead
  - Meant to change state on the server
    - Clients should ask confirmation before resubmitting

```
POST /login.php HTTP/1.0
Host: www.someplace.example
Pragma: no-cache

Cache-Control: no-cache
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.5a)
Referer: http://www.someplace.example/login.php
Content-type: application/x-www-form-urlencoded
Content-length: 49

username=jdoe&password=BritneySpears
```

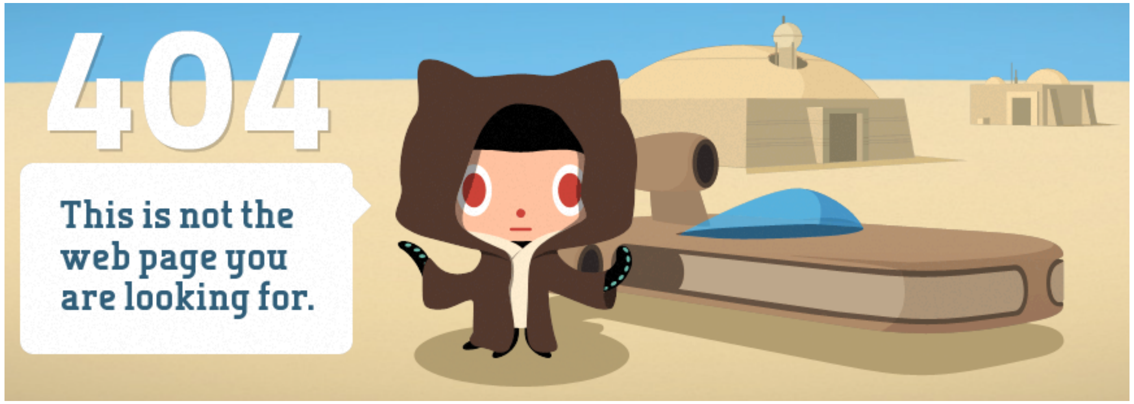
Request headers

- Main request headers
  - Host
    - Specifies target host on the server
    - Supports virtual-hosting
  - User-Agent
    - Describes browser compatibility
    - More in lecture on privacy and tracking
  - Referer
    - Where present, is the URL of where the request originated from
    - We'll see that it's important for security…
  - Cookie
    - Contains the cookies (key-value pairs) stored on behalf of the server
  - Authorization
    - Provide credentials for HTTP Basic or Digest authentication schemes
  - Accept-Encoding
    - Specifies acceptable compression methods for the HTTP response
- Like for URLs, original meaning of header may not be reflected in current use
  - Client and server can add, override, misuse headers
  - We'll see an example related to user tracking

HTTP response codes

Imperial College London

- 200 OK Success
  - The request has succeeded
  - 2xx codes are for successful requests
- 302 Found
  - The requested resource resides temporarily under a different URI
  - 3xx codes indicates that a redirection is necessary
  - In principle only GET or HEAD requested should be redirected automatically by the client
  - In practice also POST requests are redirected, but changed into GET (removing body)
- 404 Not Found
  - The server has not found anything matching the Request-URI
  - 4xx codes denote an error in the client request
- 500 Internal Server Error
  - The server encountered an unexpected condition which prevented it from fulfilling the request
  - 5xx codes denote a server error

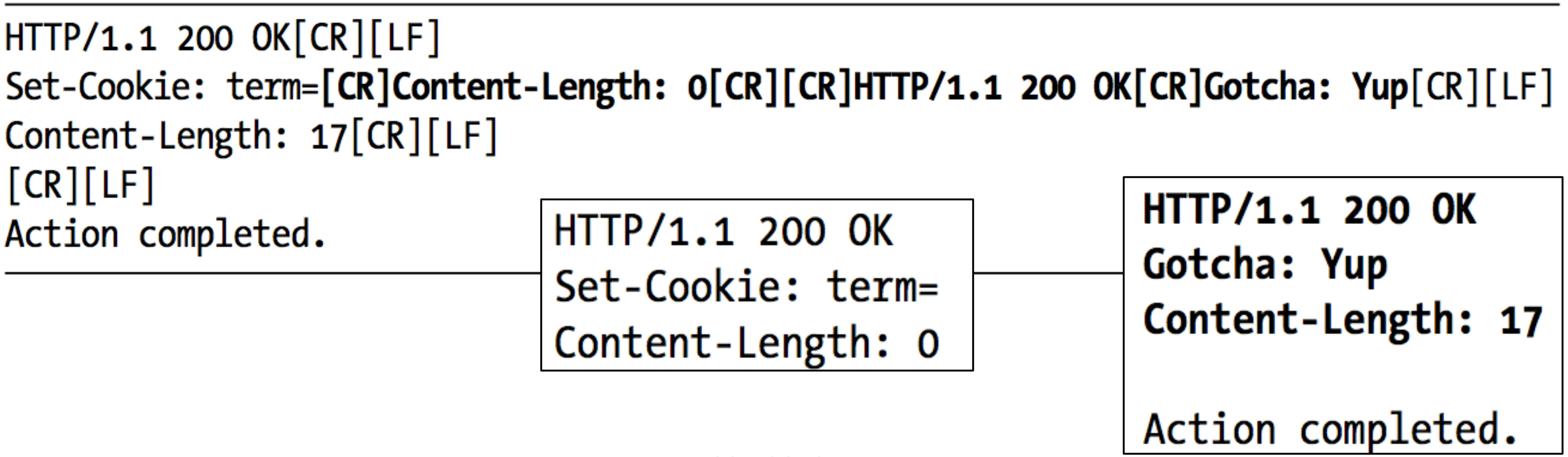


Response headers

- Main response headers
  - Content-Type
    - Specifies MIME type and character set for response
  - Location
    - Combined with 3xx response code, redirects client to different server
  - Set-Cookie
    - Requests client to store or delete some cookie on behalf of the server
  - WWW-Authenticate
    - HTTP Basic or Digest authentication schemes must be used to access resource
  - Content-Encoding
    - Specifies compression method used
  - Cache-Control
    - Specifies desired caching behaviour for client and intermediary caches
- Many more headers are currently in use
  - We'll see the security-relevant ones: CSP, CORS, HSTS, HPKP, …

HTTP security issues

- HTTP is over TCP/IP
  - No confidentiality or integrity of headers or messages against eavesdroppers or MITM
- Caching
  - If an HTTP proxy cache is poisoned, downstream clients will receive rogue HTTP responses
- Referer header
  - User visits site A and then site B
  - Privacy issue: B learns that user visited A
  - Security issue: if query string for A contained sensitive parameters, B can see them
  - Put sensitive data in the POST body, rather than in the GET query string
- Response splitting
  - Attacker could confuse client to accept bogus responses over keepalive connection



```
HTTP/1.1 200 OK[CR][LF]
Set-Cookie: term=[CR]Content-Length: 0[CR][CR]HTTP/1.1 200 OK[CR]Gotcha: Yup[CR][LF]
Content-Length: 17[CR][LF]
[CR][LF]
Action completed.
```

```
HTTP/1.1 200 OK
Set-Cookie: term=
Content-Length: 0
```

```
HTTP/1.1 200 OK
Gotcha: Yup
Content-Length: 17

Action completed.
```

HTTP versions

- HTTP/0.9: HyperText Transfer Protocol
  - Co-designed with HTML
  - By Berner Lee et al. at CERN (1989)
- HTTP/1.1 currently supported by most of the web
  - Originally specified in RFC 2616 (1999)
    - Superseded by RFCs 7230-7235 (2014)
  - Mostly backward compatible with HTTP/1.0
  - Compatibility with HTTP/0.9 introduces some issues
    - See *Tangled Web*
- HTTP/2, based on Google's SPDY
  - Approved as Proposed Standard by IESG in February 2015
    - By now most browsers support it, about 23% of websites can use it
  - Retains compatibility with HTTP 1.1
  - Adds features (mostly, it's faster)
    - Servers can push data
    - Requests are multiplexed over TCP connections, saving time to start new ones
    - Headers can be compressed
    - Some implementations use HTTP/2 only over TLS: security by default

HTTPS

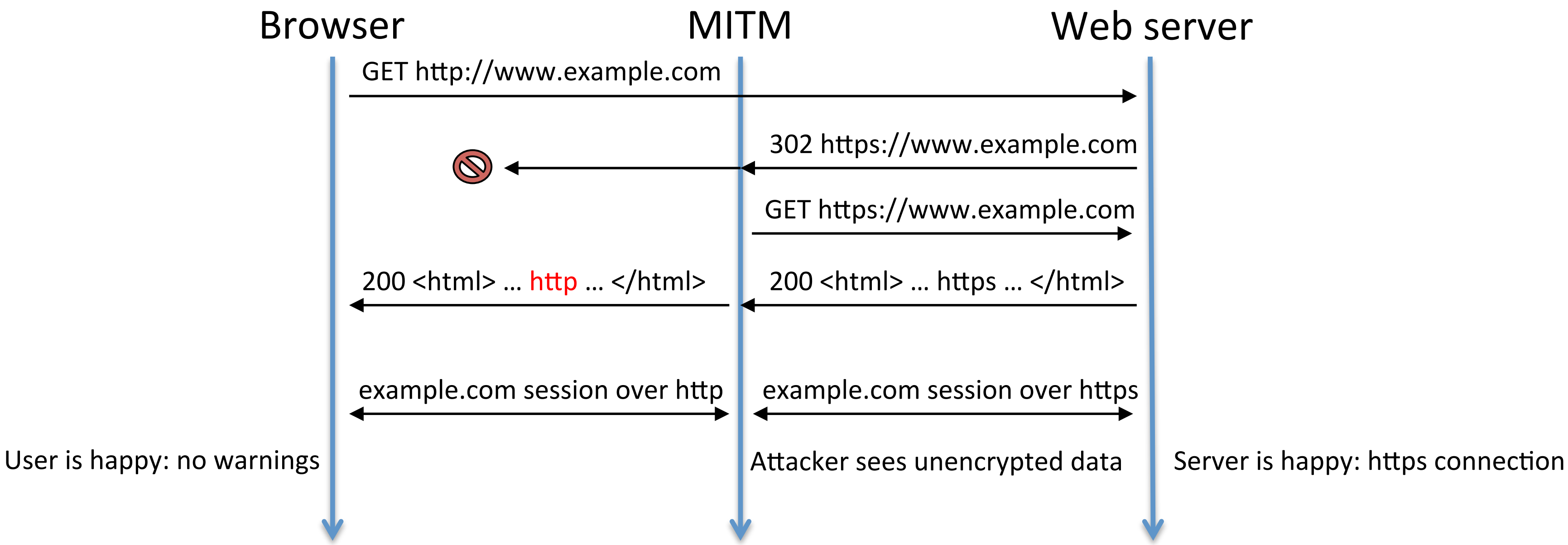
- HTTPS consists in running HTTP over an encrypted TLS connection
  - TLS provides confidentiality and integrity to the HTTP connection
  - Prevents DNS spoofing
    - Attacker should not be able to create fake certificate for spoofed domain
- HTTPS (RFC 2818) is supported by vast majority of HTTP clients
  - Yet not all traffic goes over HTTPS
    - Some cost of using public-key crypto
    - Increased latency: first request to a website is slowed down
    - ISPs cannot cache HTTPS traffic
    - Owning and maintaining certificates can be expensive
      - https://letsencrypt.org initiative provides free certs to improve uptake of HTTPS
- HTTPS runs in the browser, and a human controls the browser
  - Problems with accepting invalid certificates
  - When we talk about browser security, we will see UI attack example
- Spoofed certificates, compromised CAs invalidate TLS guarantees
  - Countermeasure: HTTP Public Key Pinning (HPKP)
    - Domain includes a response header

      ```
      Public-Key-Pins: max-age=5184000; pin-sha256="r/mIkG3eEpVdm+u/ko/
      cwxzOMo1bk4TyHIlByibiA5E="; …
      ```
    - Browser caches hash of public key for that domain
    - Time validity: too short defeats the purpose, too long prevents revocation

SSL stripping and HSTS

Imperial College London

- Unsafe to upgrade a connection from HTTP to HTTPS
  - **SSL stripping attack**:



Browser
MITM
Web server
GET http://www.example.com
302 https://www.example.com
GET https://www.example.com
200 <html> … http … </html>
200 <html> … https … </html>
example.com session over http
example.com session over https
User is happy: no warnings
Attacker sees unencrypted data
Server is happy: https connection

- Countermeasure: Strict Transport Security (HSTS)
  - HTTP response header: `Strict-Transport-Security`
  - Tells browser to load pages from that domain only over HTTPS
  - Saved for future requests depending on `max-age` = *seconds* parameter
- Bootstrapping problem: HSTS header must be sent over HTTPS
  - How to prevent SSL stripping on the first connection from HTTP?
  - Browsers have lists of websites that must be connected over HTTPS directly (this doesn't scale)
  - DANE: associate HSTS to DNSSEC