# Web security: web basics

Myrto Arapinis
School of Informatics
University of Edinburgh

March 22, 2019

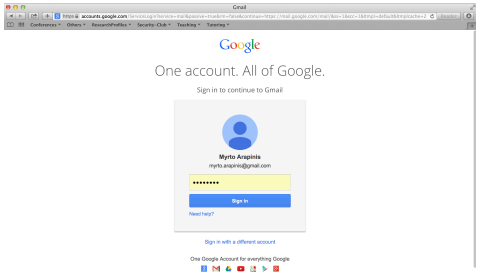
# Web applications

The web has changed the way we leave our lives:

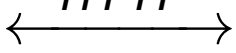
- online banking,
- online shopping,
- social networking,
- entertainment,
- education,
- news,
- ...

and has brought new classes of security and privacy concerns

# Web applications

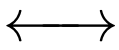



Client
(HTML, JavaScript)




Google

Server
(PHP)



Database
(SQL)

# URLs

A web browser identifies a website with a uniform resource locator (URL).

```
Protocol://host/FilePath?argt1=value1&argt2=value2
```

This naming scheme allows referring to content on distant computers in a simple and consistent manner:

- `Protocol:` protocol to access the resource (`http`, `https`, `ftp`, ...)
- `host:` domain or IP address of the server storing the resource
- `FilePath:` path to the resource on the host
- Resources can be static (`file.html`) or dynamic (`do.php`)
- URLs for dynamic content usually include arguments to pass to the process (`argt1`, `argt2`)

# HTTP requests

## GET request

```
GET HTTP/1.1
Host:  www.inf.ed.ac.uk
User-Agent:  Mozilla/5.0
             (X11; Ubuntu; Linux x86_64; rv:29.0)
             Gecko/20100101 Firefox/29.0
Accept:  text/html,application/xhtml+xml,
         application/xml;q=0.9,*/*;q=0.8
Accept-Language:  en-US,en;q=0.5
Accept-Encoding:  gzip, deflate
Connection:  keep-alive
```

- After establishing a TCP connection to the web server, the browser sends HTTP requests to that server
- HTTP requests begin with a request line (`GET` or `POST` command)
- An HTTP request consist of the headers section, and the message body

# HTTP responses

```
HTTP/1.1 200 OK
Server:  Apache
Cache-control:  private
Set-Cookie:  JSESSIONID=B7E2479EC28064DF84DF4E3DBEE9C7DF;
             Path=/
Content-Type:  text/html;charset=UTF-8
Date:  Wed, 18 Mar 2015 22:36:30 GMT
Connection:  keep-alive
Set-Cookie:  NSC_xxx.fe.bd.vl-xd=ffffffffc3a035be45525d5f4f58455e445a4
Content-Encoding:  gzip
Content-Length:  4162

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
    Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
    xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xml:lang="en" lang="en">
<head>
<title> Informatics home | School of Informatics </title>
...
```

# Hypertext Markup Language (HTML)

- The main body of a web page is encoded using HTML.
- HTML provides a structural description of a document using special tags.
- Once all the responses for a page are received, the browser interprets the delivered HTML file and displays the content.
- HTML includes a mechanism called forms to allow users to provide input to a website in the form of variables represented by name-value pairs.
- The server can then process form variables using server-side code.
- Forms can submit data either using the `GET` (name-value pairs encoded in the URL) or the `POST` method (name-value pairs encoded in the message body).

# Dynamic content

- Pages with dynamic content can change after their delivery to the client browser, eg. in response to user interaction or other conditions.
- For providing dynamic content, scripting languages such as Javascript were introduced.
- The Document Object Model (DOM) is a means for representing and accessing the content of a page.
- Scripts can alter/manipulate the content of a page by accessing/updating the DOM of the page.
- To indicate to a browser that Javascript is being used, the `<script>` and `</script>` tags:
  - Javascript allow programmers to define functions
  - Javascript includes several standard programming constructs such as `for`, `while`, `if/then/else`, ...
  - Javascript also handles events, eg. user clicks on a link, user hover mouse pointer over a portion of the page

# How is state managed in HTTP sessions

HTTP is stateless: when a client sends a request, the server sends back a response but the server does not hold any information on previous requests

**The problem:** in most web applications a client has to access various pages before completing a specific task and the client state should be kept along all those pages. How does the server know if two requests come from the same browser?
Example: the server doesn't require a user to log at each HTTP request

**The idea:** insert some token into the page when it is requested and get that token passed back with the next request

**Two main approaches** to maintain a session between a web client and a web server

- use hidden fields
- use cookies

# Hidden fields (1)

**The principle**

Include an HTML form with a hidden field containing a session ID in all the HTML pages sent to the client. This hidden field will be returned back to the server in the request.

Example: the web server can send a hidden HTML form field along with a unique session ID as follows:

```
<input type="hidden" name="sessionid" value="12345">
```

When the form is submitted, the specified name and value are automatically included in the GET or POST data.

# Hidden fields (2)

## Disadvantage of this approach

- it requires careful and tedious programming effort, as all the pages have to be dynamically generated to include this hidden field
- session ends as soon as the browser is closed

## Advantage of this approach

All browser supports HTML forms

# Cookies (1)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments

# Cookies (1)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie

# Cookies (1)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie
- Cookies are only sent back by the browser to their originating host and not any other hosts. Domain and path specify which server (and path) to return the cookie

# Cookies (1)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie
- Cookies are only sent back by the browser to their originating host and not any other hosts. Domain and path specify which server (and path) to return the cookie
- A server can set the cookie's value to uniquely identify a client. Hence, cookies are commonly used for session and user management

# Cookies (1)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie
- Cookies are only sent back by the browser to their originating host and not any other hosts. Domain and path specify which server (and path) to return the cookie
- A server can set the cookie's value to uniquely identify a client. Hence, cookies are commonly used for session and user management
- Cookies can be used to hold personalized information, or to help in on-line sales/service (e.g. shopping cart)...

# Cookies (1)

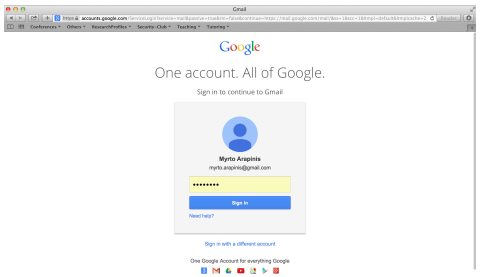
- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie
- Cookies are only sent back by the browser to their originating host and not any other hosts. Domain and path specify which server (and path) to return the cookie
- A server can set the cookie's value to uniquely identify a client. Hence, cookies are commonly used for session and user management
- Cookies can be used to hold personalized information, or to help in on-line sales/service (e.g. shopping cart)...

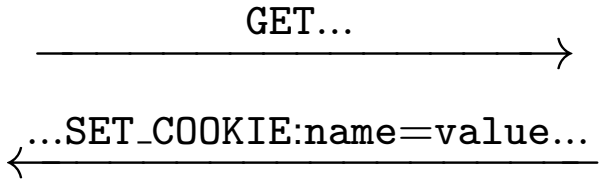
**Main limitation**

Users may disable cookies in their browser

# Cookies (2)

Cookies are set on the client's system when the server uses the `Set-Cookie` field in the HTTP header of its response:




Google

A cookie has several attributes:

```
Set-Cookie:  name=value[; expires=date]
        [; domain=dom][; path=p][; Secure][; HttpOnly]
  expires : (whentobedeleted)
  domain : (whentosend)  } scope
  path : (whentosend)    }
  Secure : (onlyoverSSL)
  HttpOnly : (onlyoverHTTP)
```

# Cookies (3)

- A cookie is valid for the domain it is set for, and all its subdomains.
- A subdomain can set a cookie for a higher-level domain but not vice-versa.
  `mail.example.com` can access cookies set for `example.com`
  `example.com` cannot access cookies set for `mail.example.com`
- Hosts can access cookies set for their top level domains, but hosts can only set cookies one level up in the domain hierarchy.
  `one.mail.example.com` can access cookies set for `example.com`
  `one.mail.example.com` cannot set cookies for `example.com`
- A website can only set a cookie for a domain that matched the domain of the HTTP response.
- `Http-Only`: if enabled scripting languages cannot accessing or manipulating the cookie.

# Web security: security goals

# Security goals

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

1. visiting `evil.com` should not infect my computer with malware, or read and write files
   **Defenses:** Javascript sandboxed, avoid bugs in browser code, privilege separation, *etc*

2. visiting `evil.com` should not compromise my sessions with `gmail.com`
   **Defenses:** same-origin policy – each website is isolated from all other websites

3. sensitive data stored on `gmail.com` should be protected

# Threat model

**Web attacker**

- controls `evil.com`
- has valid SSL/TLS certificates for `evil.com`
- victim user visits `evil.com`

**Network attacker**

- controls the whole network: can intercept, craft, send messages

A Web attacker is weaker than a Network attacker