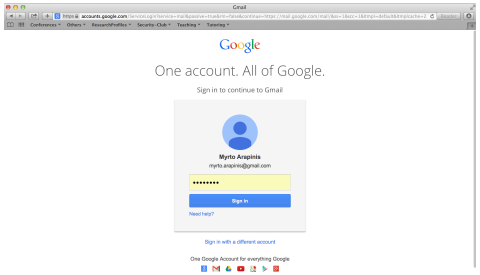
Web security: web basics

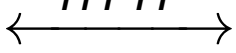
Myrto Arapinis
School of Informatics
University of Edinburgh

November 13, 2017

Web applications

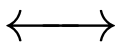


Client
(HTML, JavaScript)



Google

Server
(PHP)



Database
(SQL)

URLs

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

- `Protocol:` protocol to access the resource (`http`, `https`, `ftp`, ...)
- `host:` name or IP address of the computer the resource is on
- `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
```

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>
...
```

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)...

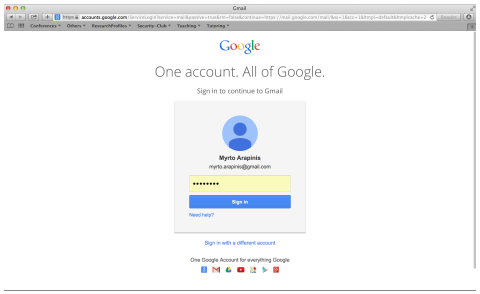
Cookies (1)

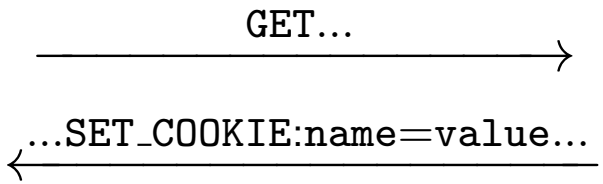
- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent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





Google

A cookie has several attributes:

```
Set-Cookie:  value[; expires=date][; domain=domain]
                   [; path=path][; secure][; HttpOnly]
```

- expires : (whentobedeleted)
- domain : (whentosend) } scope
- path : (whentosend) } scope
- secure : (onlyoverSSL)
- HttpOnly : (onlyoverHTTP)

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

OWASP TOP 10 Web security flaws (2013)

A1 – Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2 – Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3 – Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A4 – Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5 – Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
A6 – Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A7 – Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
A8 - Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
A9 - Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.
A10 – Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.