Browser Security

In the early days, browsers supported HTML but no executable content. Two independent approaches were developed to add executable content, thereby making it easier to provide better customization and animation.

- *Java sandbox model*
  - goal: allow arbitrary apps from internet to be downloaded and run
  - browser establishes a policy sufficiently restrictive to run arbitrary code without problems, then leaves the enforcement up to the JVM
    - no access to filesystem, etc.
    - least possible privileges to libraries to avoid untrusted code exploiting trusted code
  - could run "flying pigs" demo, but not many useful things due to lack of access to local resources
  - subsequently, such visualization functions have been taken over by Flash player.
- *Javascript model*
  - higher level language that exposes certain objects and methods provided by the browser (access local resources through browser as interface, rather than directly)
  - more appropriate model than Java sandbox model
  - e.g. can do file upload, but it must be done through a trusted function in the browser, gets user's consent
  - still need to make sure that certain security policies are enforced
  - websites shouldn't be able to influence user's interaction with other websites
  - "Same Origin" policy – e.g. script on yahoo.com can use yahoo.com cookies, but not google.com cookies.
  - original design was not comprehensive enough in terms of security since the security landscape was relatively simple at the time

Security Issues They Accounted For

(a) Confidentiality of data handled by the Browser

- e.g. if you are from site XYZ, you can't access cookies from site ABC

(b) Browser and host integrity

- attacks that modify browser data structures or host resources in ways that might compromise their security

Issues They Didn't Account For

(c) Threats from truly malicious servers

- integrity of other hosts
  - Cross-site Request Forgery (CSRF)

        - e.g. suppose you can log on to a router from your local network, and router account info is default (or easy to guess). Malicious javascript can forge request from you to the internal router and attack the router via HTTP
        - very common attack vector
    - DDoS on third-party websites
        - Paper on this threat “Puppetnets”
- ability to run code locally creates a powerful adversary
    - Javascript can be used to set stage for heap corruption attacks
    - Enables attacks that involve searching for vulnerabilities (that cannot be searched for from the outside)
- misplaced trust can cause grave damage
    - Cross-Site Scripting (XSS)