Web Application Security - Contd.

Information Security – Lecture 15

Aadil Zia Khan

MIME Sniffing

- MIME-sniffing is a browser feature
- Browser auto-detects the Content-Type of a resource that it downloaded and fixes it
- For example
  - Webpage asks the browser to render an image at /fancypic.gif, but the server sets the wrong type when serving it to the browser (i.e., Content-Type: text/plain)
  - Previously the browser would fail to display the image properly – however, a fix allowed the browsers to sniff the content type and ignore the server's Content-Type header in case of a mismatch
  - Problem???

MIME Sniffing Attack

- Suppose a website allows users to upload images
- The user uploads a /malicious.jpg file that in reality is a JavaScript code file
- Once the file is uploaded, the website would include it in its own HTML
- When the browser tries to render the image, it would detect that it's a script instead, and execute it on the victim's browser

MIME Sniffing Attack - Defense

- Server sets the X-Content-Type-Options: nosniff header
- It tells the browser to disable MIME-sniffing
    - If some file has a mismatch in terms of type and content – it shouldn't try to guess things and simply not load that file

Feature-Policy

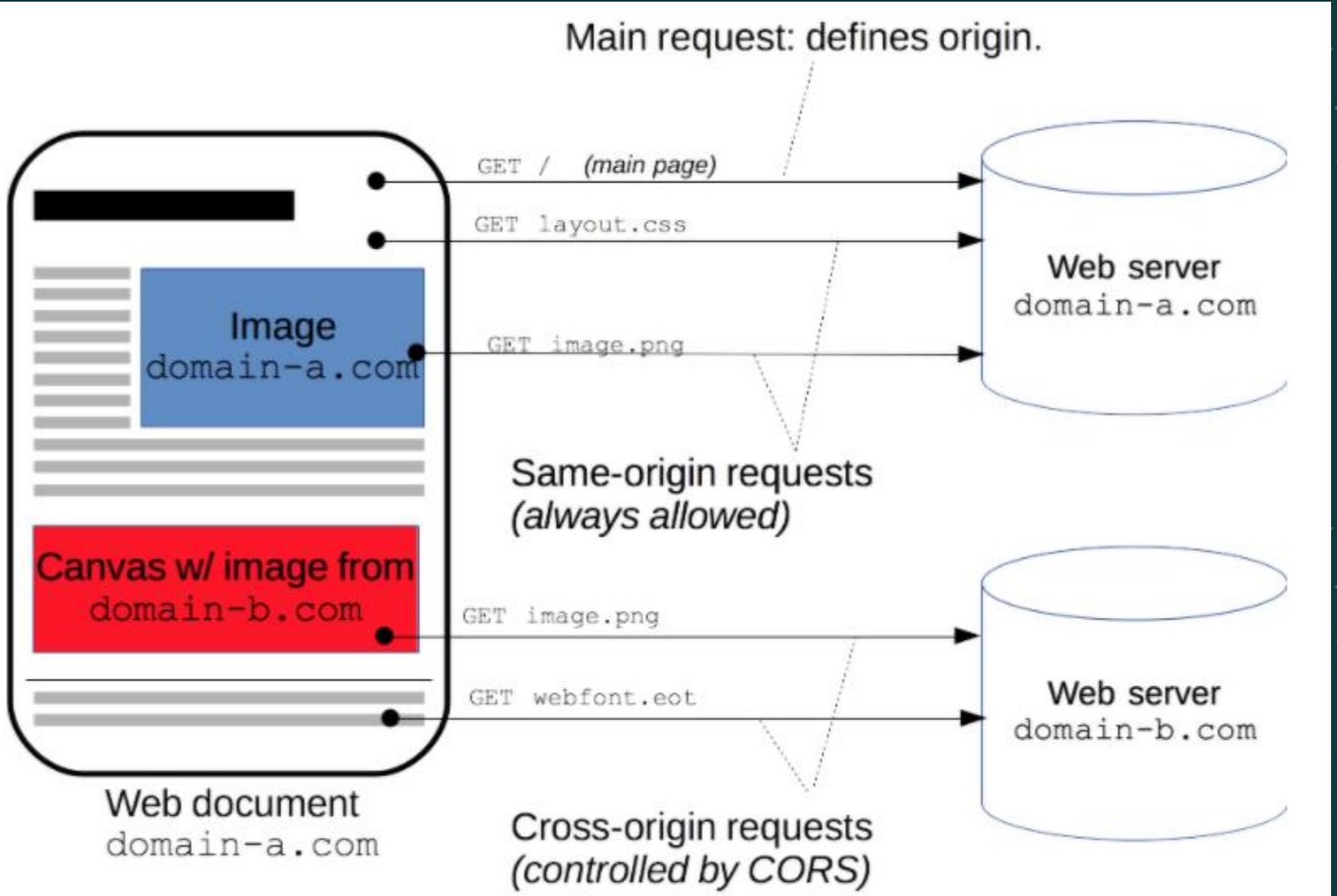
- Server uses the Feature-Policy header to inform the browser about the enabled features within the current page
- Example Usage
  - Feature-Policy: vibrate 'self'; camera 'none‘
  - This setting informs the browser that the current page (and nested iframes from the same origin) cant use the camera but they can use vibration

Cross Origin Resource Sharing

- Safe browser behavior: HTTP requests can only be triggered across the same origin through JavaScript
  - Browser contains useful information for an attacker – e.g., cookies and tokens
  - Suppose attacker sets up a malicious page fancyanimation.com that triggers a request to nationalbank.pk
  - If victim is logged in on the bank's website, the attacker would be able to execute HTTP requests with your credentials - potentially conducting bank transactions

Cross Origin Resource Sharing

- Cross origin requests become necessary in some scenarios
  - For example, what if a webpage (travel bookings) wants to load data from a third party web service (weather data)
- Browsers implement Cross Origin Resource Sharing (CORS) to control cross-domain requests
- CORS is an HTTP-header based mechanism that allows a server to indicate domains other than its own from which a browser should permit loading of resources
  - CORS also relies on a mechanism by which browsers make a “preflight” request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request
    - Prefetch ensures that the browser doesn’t send an HTTP POST message (containing private data) to the cross origin server (e.g., a bank) without asking its permission


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image
domain-a.com
Web server
domain-a.com
Same-origin requests
(always allowed)
Canvas w/ image from
domain-b.com
GET image.png
GET webfont.eot
Web server
domain-b.com
Web document
domain-a.com
Cross-origin requests
(controlled by CORS)

Cross Origin Resource Sharing - Defense

- Server sends an Access-Control-Allow-Origin header to tell the browser that it can receive resources from other origins
- Example
  - Access-Control-Allow-Origin: *
    - Allows resources from any origin
  - Access-Control-Allow-Origin: https://good.com
    - Server can restrict the browser to the specified URLs

Reporting API

- Reporting API allows a webserver to advertise to the browser a particular URL on which it can receive reports
  - The Report-To HTTP header is used to specify the reporting URL
- Reporting API can be used to receive information on various aspects of user experience
  - CSP and feature-policy violations
  - Browser blocking risky code
  - When the webservice uses a deprecated API
  - When the client crashes