Web App Security
Browser Isolation
Mendel Rosenblum
What could go wrong with our web app?

- Our app could allow an attacker to view and/or modify any information or perform any operations we provide
  - Leak information provided
  - Perform actions on behalf of the user

- Our app could be used to attack anything on our user's machine and or anything our user machine can talk to
  - If the user trusts us we can allow damage far beyond what the user gives to us
Security is an hard problem

- Many opportunities for attackers
  - Full stack means there are many interface that an attacker can use

- Hard to identify all the vulnerabilities
  - Complexity of system make it impossible guarantee no vulnerabilities

- Even a small mistake can compromise entire application
  - Only as strongest as the weakest link
Modes of attack

- Attack the connection between browser and web server
  - Steal password
  - Hijack existing connection

- Attack the server
  - Inject code that does bad things

- Attack the browser
  - Inject code that does bad things

- Breach the browser, attack the client machine

- Fool the user (phishing)
Security Defence

- Isolation in browsers
  - Web app run in isolated sandbox

- Cryptography
  - Protect information from unauthorized viewing
  - Detect changes
  - Determine origin of information

- Web development frameworks
  - Use patterns that help, avoid dangerous ones
Challenge of isolation in the browser

- Web content comes from many sources, not all equally trusted
- Trusted and untrusted content are in close proximity
  - Frames, tabs, sequential visits
- Must separate various forms of content so that untrusted content cannot corrupt/misuse trusted content
Example: a "good" page displays a sponsored ad

- Attackers can buy advertisements, use them to attack good pages.
- Advertiser gets to supply content for ad
  - "good" page links to advertiser site in <iframe>
- Ad can contain <script> elements that access DOM, submit forms, etc.
  - parent.frames[0].forms[0].submit;
Same-Origin Policy

- General idea: separate content with different trust levels into different frames, restrict communication between frames
- One frame can access content in another frame only if they both came from the same origin
- Origin is
  - Protocol
  - Domain name
  - Port
- Access applies to DOM resource, cookies, XMLHttpRequest/AJAX requests
- Doesn't apply: `<script>` tags
  - Javascript executes with full privileges of the enclosing frame.
same-origin policy is too restrictive

- There are times when it is useful for frames with different origins to communicate
  - Example: Sub-domains of same organization
  - Web fonts
  - Content distribution network

- Browsers allows page to set its domain with `document.domain`
  ```javascript
  document.domain = "company.com";
  ```

- Limited to sub-domain sharing
HTML5 feature: Access-Control-Allow-Origin

- Access-Control-Allow-Origin header in HTTP response:
  
  Access-Control-Allow-Origin: http://foo.com
  Access-Control-Allow-Methods: PUT, DELETE

- Specifies one or more domains that may access this object's DOM.
  Can use "*" to allow universal access.
HTML5 postMessage - safe messaging

- Sender (from domain a.com) to an embedded frame of different domain:
  ```javascript
  frames[0].postMessage("Hello world", "http://b.com/");
  ```
- Receiver (domain b.com) can check origin:
  ```javascript
  window.addEventListener("message", doEvent);
  function doEvent(e) {
    if (e.origin == "http://a.com") {
      ... e.data ...
    }
  }
  ```
Cookie Security

- Cookies can be read and written from Javascript:

  ```javascript
  alert(document.cookie);
  document.cookie = "name=value; expires=1/1/2011"
  ```

- Browsers use the same-origin policy to restrict access to cookies.