Simple attack
<script>
</script>

Attack detected by user

Stealthy attack
<script>
img = new Image();
</script>
Cookie flags: HttpOnly & Secure

- **HttpOnly** is an *additional flag* included in a Set-Cookie HTTP response header
- **HttpOnly** works if supported by the browser
- This prevents cookies from being accessed by a client-side script, protecting them from XSS
- **Secure flag** sends cookies when the request is going to a HTTPS page
Client-Side Controls

• Web security problems arises because clients can submit arbitrary input

• What about using client side controls to check the input?

• Which kind of controls?
Client-Side Controls

• A standard application may rely on client-side controls to restrict user input in two general ways:
  • Transmitting data via the client component using a mechanism that should prevent the user from modifying that data
  • Implementing measures on the client side, with the aim of adding a control layer before user input is submitted
Bypassing Web Client-Side Controls

• In general a security flaw because it is easy to bypass

• The user:
  – Has a full control over the client and the data it submits
  – Can bypass any controls that are client-side and not replicated on the server

• Why these controls are still useful?
  – E.g. for load balancing or usability
  – Often we can suppose that the vast majority of users are honest
Transmitting Data Via the Client

- A common developer bad habit is passing data to the client in a form that the end user cannot directly see or modify.

- Why is it so common?
  - It removes or reduces the amount of data to store server side per session.
  - In a multi-server application it removes the need to synchronize the session data among different servers.
  - The use of third-party components on the server may be difficult or impossible to integrate.

- Transmitting data via the client is often the easy solution but unfortunately is not secure.
Common Mechanisms

• **HTML Hidden fields**
  – A field flagged hidden is not displayed on-screen

• **HTTP Cookies**
  – Not displayed on-screen, and the user cannot modify directly

• **Referer Header**
  – An optional field in the http request that it indicates the URL of the page from which the current request originated

• If you use the proper tool you can tamper the data on the client-side
Web client tool

• Web inspection tool:
  – Firefox Firebug or Chrome web developer:
    • powerful tools that allow you to edit HTML, CSS and view the coding behind any website: CSS, HTML, DOM and JavaScript

• Web Proxy:
  – Burp, OWASP ZAP, etc.
    • Allow to modify GET or POST requests
Demos

• Owasp Webgoat 7, parameter tampering
  – Bypass html field restrictions
  – Exploit hidden fields
  – Bypass client side java script validation
Same Origin Policy (SOP)

• The SOP restricts how a document or script loaded from one origin can interact with a resource from another origin

• One origin is permitted to send information to another origin, but one origin is not permitted to receive information from another origin

• Without "allow sending", there would be no "web" at all because each origin would be allowed to link only to itself
Browser add-ons: Request Policy (RP)

• RP gives you a default deny policy for cross-site requests

• Cross-site requests are:
  – requests that your browser is told to make by a website you are visiting to a completely different website

• RP allows you to whitelist cross-site requests you trust
Cross-Site Request Forgery (CSRF)

- Malicious website has script that redirects and issues a request on target website
- The request is executed by the target website on behalf of the user
  - E.g., funds are transferred from the user to the attacker
CSRF defense

• **Synchronizer token pattern**
  – A token is embedded by the web application in all HTML forms and verified on the server side
  – CSRF request rejected because attacker cannot guess ID

• **Cookie-to-Header Token**
  – On login, web site sets a CSRF cookie containing a random token
  – A script operating on the client side reads its value and copies it into a custom HTTP header sent with each transactional request
  – Security is based on the assumption that only a Script running within the same origin will be able to read the cookie's value and then to write in the header
Clickjacking involves **clicking** the user into filling out forms data.

This allows data to be **submitted** across domains that bypasses Cross-Site Request Forgery defenses.

It can involve **clicking buttons**, or using the **drag-and-drop** API in HTML5 to fill in text.

– `<div draggable="true" ondragstart="event.dataTransfer.setData(‘text/plain’, ‘Evil data’)"><h3>DRAG ME!!</h3></div>`
NoScript from noscript.net

- **Extension** for Firefox, similar for Chrome
- **Prevents** execution of JavaScript, Java, and Flash – the most common vectors for web-based attacks
- Allows trusted sites **interactively**
- Often able to view content of interest without enabling all scripts (selective permissions)
- It is **not easy** to use and takes some time to understand which features can be allowed
This Site is Secure?

• Web Technology introduced a new range of vulnerabilities not present in earlier Client Applications

• Unfortunately it is not enough to say:
  – We use Https or SSL -128 bit so this site is absolutely secure

• Main problem, users can:
  – Submit arbitrary inputs
  – Interfere with any piece of data transmitted
  – Forge requests or parameters with tools external to the browser

• In the modern web application frameworks (e.g. Spring, Struts, Hibernate, etc.) or CMS (e.g. wordpress, joomla, etc.) are commonly used and they can have bugs.