<https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet>

* Hash the password to reduce password size
  + resolves performance bottleneck for very large passwords
* Use a cryptographically strong and credential-specific salt
  + salt: a fixed-length cryptographically strong random value
  + append salt to credential data as input to protective function
    - [protected form] = [salt] + protect([protection func], [salt] + [credential]);
  + pros of salt:
    - prevent the protected form from revealing two identical credentials
    - argument entropy fed to protecting function without relying on credential complexity
  + good practice:
    - generate a unique salt upon creation of each stored credential
    - use cryptographically-strong random data as salt

Storage, transportation, autofill mechanism?

**PM autofill threats:**

* automatic autofill vs. manual autofill, both can be problematic problematic
  + manual autofill > automatic autofill
  + manual autofill: user might not be sensitive to potential attack, e.g. form action is not visible to user
* pages within same domain,
  + attacker can attack least-secure page
  + different sites on same domain treated as one
* same page on different protocol
  + dangerous
* modified form action
  + loaded form action points to correct URL, but changed by JavaScript
* website autocomplete attribute
  + lots of PM does not respect autocomplete attribute, e.g. still autofill when autocomplete attribute = “off”
  + this attribute may be modified by attacker who controls the network
* broken HTTPS behavior
  + autofill can lead to significant attacks
* different field name
* visibility = None vs opacity = 0
* synchronization
  + security of PM bounded by weakest PM it synchs with

Solutions:

* Forcing user interaction
* when autofill, make password unreadable to JavaScript
* abort autofill when username / password field modified
* check form action before running JavaScript

Standardize practice for server

* Use HTTPS on both the login page and page it submits to. Ideally, use HTTPS everywhere on the site and enable HSTS (HTTP Strict Transport Security) to prevent pages from ever loading under HTTP.
* Use CSP (Content Security Policy) to prevent the execution of inline scripts, making the injection of JavaScript directly into the login page ineffective
* Host the login page in a different subdomain that the rest of the site (i.e., login.site.com instead of site.com). This limits the number of pages considered same-origin with the login page, reducing the attack surface

active mixed content, XSS cross site scripting

Usability concerns: PM not save / autofill passwords will not be popular

iFrame is susceptible to attacks

HTTP vs HTTPS, certificate

Wifi encryption protocol: WPA2

Web storage:

* local storage:
  + per origin, the data is available to all scripts loaded from pages from the same origin that previously stored the data
* session storage:
  + per-origin-per-window-or-tab and is limited to the lifetime of the window

**Password Strength**

P. G. Kelley et al., "Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms," 2012 IEEE Symposium on Security and Privacy, San Francisco, CA, 2012, pp. 523-537.  
doi: 10.1109/SP.2012.38  
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6234434&isnumber=6234400>

- basic 16 out performs comprehensive 8

M. Dell' Amico, P. Michiardi and Y. Roudier, "Password Strength: An Empirical Analysis," 2010 Proceedings IEEE INFOCOM, San Diego, CA, 2010, pp. 1-9.  
doi: 10.1109/INFCOM.2010.5461951  
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5461951&isnumber=5461899>

<https://nulab-inc.com/blog/nulab/password-strength/>

**Password Generator design:**

- minimum length of 16

- include special character

- not include dictionary words or inverted words

- no continuous sequence

- no L33t

- no repeat patterns

- no date, year, month