Chapter 2 - Core defence Mechanisms

Core elements of defence mechanisms:

- Handling user access
- Handling user input
- Handling attackers (how the app is dealt with when being directly attacked)
- Managing the application itself by enabling admins to monitor activites

Handling User access

Authentication

- treating all users as anonymous is the lowest possible level of trust
- conventional authentication model = username + password
- security-critical applications (eg banks) = conventional auth model + (additional credentials | multistage login)
- even higher security = security-critical + (client certificates | smartcards | challenge-response tokens)
- Authentication requires supporting functionality, self-registration, account recovery, password changing

Session Management

- To deal with a countless amount of simultaneous authenticated requests sites use session tokens to keep track
- Ways of dealing with tokens: cookies, hidden form fields, url query string

Access control

- Probing for these vulnerabilities is often laboruous, effort is a worthwhile investment

Approaches to input Handling

“Reject known bad”

- Blacklist-based filters can be bypassed in a stupid way
  - if `SELECT` is blocked, try `SelEcT`
  - If `or 1=1 -- -` is blocked try `or 2=2 -- -`
  - if `alert("xss")` is blocked try `prompt("xss")`
- filters designed to block specific phrases/keywords can be bypassed as such
  - `SELECT/*foo*/username,password/*foo*/FROM/*foo*/users`
  - `<img%09onerror=alert(1) src=a>`
- many blacklist filters are vulnerable to NULL byte attacks because the filter sees them as terminators
  - `%00<img src=x onerror=alert()>`

“Accept known good”

- Sometimes only matches at the start
- Sometimes regular inputs don't match what is “good” eg names contain `-` or `'`
- Can only be used well in specific circumstances, thus not used often

Sanitziation

- Potentially bad data is remove
- Potentially bad data is “escaped” by encoding

Multistep validation and canonicalization

- `<scr<script>ipt>` → `<script>`
- Canonicalization is the process of converting or decoding data into a common character set. eg `»` is converted to `>` etc

- another example is double url decoding