Best Practices - Security

# Ideas

## Good: Whitelist => Bad: Default Permit

Do not try to enumerate badness/vunerabilities and blacklist them, since default permit with blacklist design choice leads to infinite security holes and a constant race to keep up the blacklist. Instead take the time to decide what is allowed and prevent anything else. For example, whitelist html and css instead of blacklisting potentially bad design patterns.

Clear sign is if the software requires updating on a regular basis with security patches.

## Good: Design Secure => Bad: Penetrate and Patch

Penetration testing and fixing doesn’t make code design any better, it simply makes it toughened by trail and error.

'Unless your system was supposed to be hackable then it shouldn’t be hackable'

Networks should be designed to be permeable only in certain directions and only to certain traffic destined to carefully configured servers running secured software. Flaw handling should be considered in design.

## Good: Learn Secure Engineering => Bad: Learn Exploits

Fine line between learning hacking exploits and learning good engineering which prevents the exploits ever being able to be used. Hacking is not cool, instead learn to design systems which are secure by default and can't be hacked. Albeit you need to know a system design from the ground up to prevent hacking, so before this stage learning typical vunerabilities and why they work is useful in understanding and designing a better system.

## Good: Prevent Problems => Bad: Educate Users

Instead of educating users, which most of the time wont work effectively, look at the root cause of the problem and prevent or mitigate it. For example, with emails and malicous attachements. Instead of educating users to not open suspicous attachements, which they may or may not remember, quarantine attachments on delivery, filter with whitelist, and replace with links to the data stored on an isolated system, and make a user visit the system to open the file.

## Good: Pause and Think => Bad: Early Adopter

Regarding security it is unlikely that being an early adopter of a new techonolgy will lead to a secure system, since the flaws just haven't been realised yet. Instead wait a few years and use industry knowledge to create a secure system. If you have to be an early adopter, pause and think about the implications about each design decision, test it operationally, and truly understand the tech before deploying.

Can be good to go the conferences for the new tech and get a low down from someone who has used it.

# Critical System Design

## Component System Design (Bottom Up)

In engineering, design a system from the bottom up so each section of the system is throuoghly understood and limitations known:

* Understand material properties, operating ranges, and limitations
* Create larger component parts such as bearings, deficiencies and design errors are noted and corrected or mitigated with testing
* Design whole system of interconnected parts with good understanding of opertating ranges of all components

The principle that is followed is that all the verification is not an aspect of program safety, it is merely a test of that safety, in a non-catastrophic verification.