Chapter 10: All Input is Evil

1. Trust no data until you are certain it has been validated
2. Data must be validated *as it crosses* the boundary between untrusted and trusted environments
    a. Trusted: you or an entity you explicitly trust has complete control over the data
    b. Validate for security and for robustness (legitimate user error)
    c. If data is checked as it crosses the boundary, the rest of your app can run without any performance issues if data is trusted – a "little" extra code can go a long way
3. "It's difficult to find a system less reliably responsive than a hacked system"
4. Fundamental issue: in networked world, server cannot trust data from client
    a. Client requests can be forged
    b. Vice-versa also true: client cannot trust data from server
5. Strategies to defend against input attacks
    a. Define a trust boundary
    b. Create an input chokepoint
        i. Can have more than one
        ii. Consider all forms of input to program
            1. web: cross-site scripting
            2. registry
            3. files
            4. user input
            5. database
            6. open socket
    c. Any reusable component or externally reachable routine you create should screen input
6. How to validate
    a. Always check for what you consider to be valid data
        i. it's better to have a few unhappy good users because you missed something legitimate than one happy attacker!
        ii. If you look for the bad stuff you are bound to miss something
    b. Be aware of canonical representation issues (escape characters, other character sets, etc.)
7. Regular expressions as an input validation tool
    a. Most modern languages have support for regular expressions in some form
        i. C: regex.h
        ii. Java: Pattern
        iii. C++: boost libraries, STL
            1. great article: http://linuxgazette.net/issue27/mueller.html
        iv. Perl (fabulous for REs)
        v. C#: Regex
        vi. Etc.
    b. Common RE elements

`^`	Match start of string
`$`	Match end of string
`*`	Match preceding pattern 0 or more times
`+`	“ “ 1 or more times
`?`	“ “ 0 or one time
`{n}`	Match preceding pattern exactly n times
`{n,}`	“ “ n or more times
`{,m}`	“ “ no more than m times
`{n,m}`	“ “ from n through m times
`.`	Any single char other than \n
`aa | bb`	Matches aa or bb
`[abc]`	Matches any one of enclosed chars
`[^abc]`	Matches any char not in list
`[a-z]`	Matches any chars in range
`\d`	Match digit
`\D`	Match non-digit
`\n, \r, \f, \t, \v`	Formatting chars
`\s`	Matches whitespace
`\S`	Matches non-whitespace

c. Examples

`[a-fA-F0-9]+`	Match 1 or more hex digits
`<(.*)>.*<\/\1>`	Match an HTML tag (.*) remembers first tag, 1 refers to the remembered tag – variable
`\d{5}(-\d{4})?`	Zip code
`^\w{1,32}(?:\.\w{0,4})?$`	Filename check

d. Watch for escape characters and things that can slip through your RE – try and specify what’s good

e. REs and Unicode: Unicode is 16 bits – many RE tools work with 8-bit chars
   i. Know which your language handles
   ii. Link to Unicode and REs: http://www.unicode.org/reports/tr18/
   iii. Unicode categories of RE symbols:
      1. L: letters
      2. M: marks (accents, umlauts, vowel signs, enclosing marks)
      3. N: numbers
      4. P: punctuation
      5. S: symbols (math, currency, circumflex, grave, copyright, Celsius)
      6. Z: separators (space, line, paragraph)
      7. O, C: others (control codes, format characters, invisible characters, high and low surrogate characters, etc.)