Keysmith

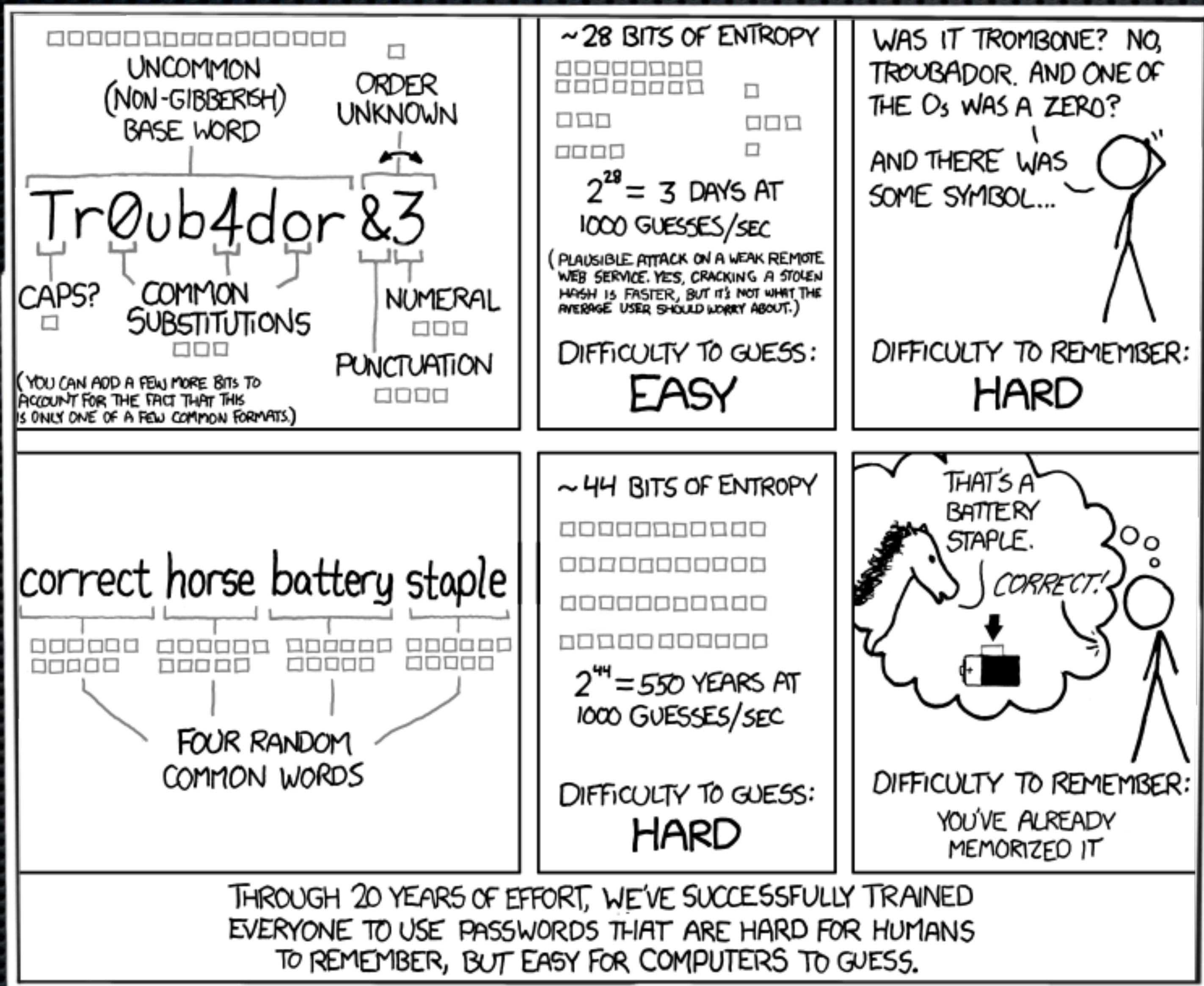
Authentication Token Generator

David Tucker

Sentinel

- tiered account manager
  - Groups of related keys are assigned a tier.
    - e.g. Bank Accounts: tier 2
  - Tiers specify key generation algorithm and expirations.
    - e.g. tier 2: concatenate 3 random words (good for 90 days)
  - Secure keys are generated automatically.
    - Keysmith

XKCD



UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS:
EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER:
HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS:
HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER:
YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Why is this way best?

Say Bob uses a 10,000 word dictionary and a true random number generator to derive a passphrase of 4 concatenated words. Let's also assume Mallory (who knows how Bob's key was created) wants to break this key and has the ability to guess 2,000 keys per second.

How long (at most) will it take Mallory to brute force Bob's passphrase?

Odds of a correct guess:

$$\frac{1}{10^4} \cdot \frac{1}{10^4} \cdot \frac{1}{10^4} \cdot \frac{1}{10^4} = \frac{1}{10^{16}}$$

Time to get the correct key:

$$\left(\frac{10^{16} guesses}{1}\right)\left(\frac{1 second}{2,000 guesses}\right) = 5(10^{12}) seconds$$

$$\approx 158440 years$$

Bottom line:

Most dictionaries have far more than 10,000 words, and most attackers will not know your word list or even your key generation algorithm!

Random Numbers

- atmospheric noise
- HTTP API
  - plain-text responses
- ranges
- example query
  - http://www.random.org/integers/?num=3&min=0&max=43238&col=1&base=10&format=plain&rnd=new

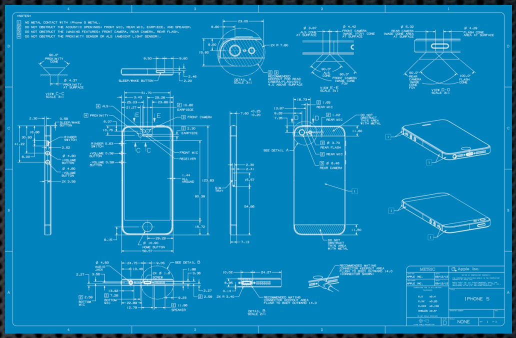
Implementation

- Imperative
  - C
- Object-oriented
  - Java
- Functional
  - Haskell
- Logic
  - ~~Prolog~~

C

- error-checking, and a lot of it
- native command-line parser
  - int getopt (int argc , char * const argv[] , const char * optstring);
- native HTTP support
  - cURL
- no simple string manipulation
- generally unforgiving

Haskell



- what memory?
  - implementation specific (due to emphasized abstraction)
- native command-line parser
  - System.Console.GetOpt
- Hackage & Cabal
  - Network.HTTP, RandomDotOrg, etc.
- generally very forgiving
- excellent documentation
  - Hoogle, #haskell, more

Java

- container-based
- requires more meta-knowledge
  - e.g. public void java.lang.System.out.println()
- no native command-line parser (more graphical)
  - JCommander
- native support for CSRNG, kinda
  - java.security.SecureRandom
- heavyweight
  - setters & getters

Questions