SCOUG

Miscellany

Greg Smith

January 21, 2017

Some Comments About Passwords

A server asking your password should *not* have your password saved anywhere in plain text. At the very least, it should be salted and hashed.

From `http://mathworld.wolfram.com/HashFunction.html`

Definition: A hash function H projects a value from a set with many (or even an infinite number of) members to a value from a set with a fixed number of (fewer) members. Hash functions are not reversible.

Common hash functions include:

- MD5 $\longrightarrow$ maps a message to 128 bits
- SHA1 $\longrightarrow$ maps a message to 160 bits
- SHA256 $\longrightarrow$ maps a message to 256 bits
- SHA512 $\longrightarrow$ maps a message to 512 bits
- Tiger $\longrightarrow$ maps a message to 192 bits
- Whirlpool $\longrightarrow$ maps a message to 512 bits

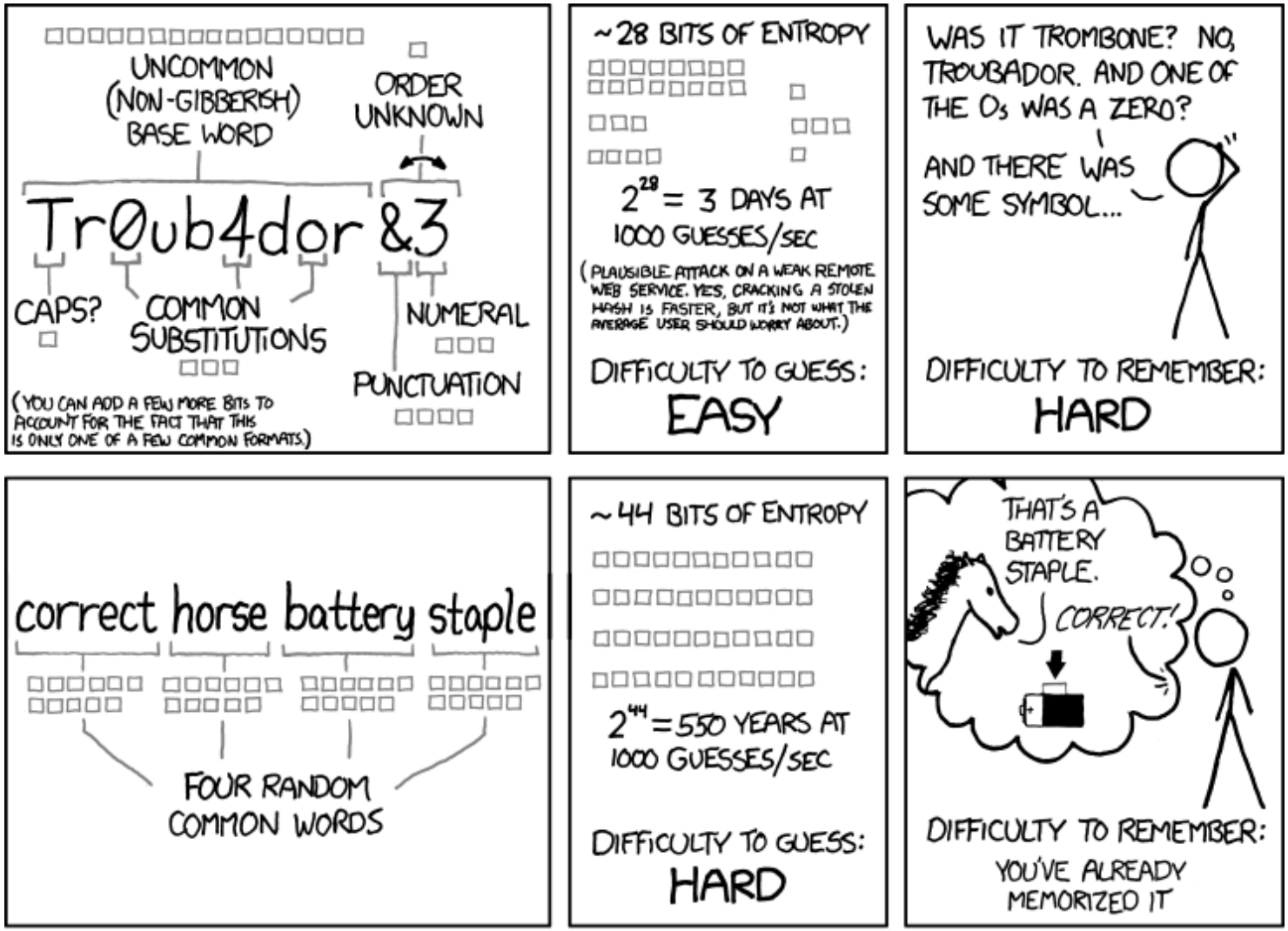
Verifying a File

Hash functions are great for verifying files since they are coded to run quickly. But quick is not good for passwords.

So many ways of storing passwords run many hashes before storing the password.

LastPass, for example, runs several thousand rounds of PBKDF2 SHA-256 salted hashes before storing your master password.

XKCD on Passwords



Passwords Are Made of Tokens

Cracking passwords cycles through sets of tokens, hashing them and comparing them to a stolen password file that has the hashed and salted passwords.

The size of the token set determines how easy it will be to crack with a random search. Consider a simple password made of three tokens.

Consider a simple token set: { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Cracking is easy with only 1000 combinations to check.

Token Set Size

A larger set of tokens is the numbers and letters

$$[0\text{-}9a\text{-}zA\text{-}Z]$$

This set now needs to cycle through $62^3 = 238{,}328$ combinations to check all of the possibilities.

A larger set would be the 1000 words in the Thing Explainer dictionary used by Randall Munroe.

This set now needs to cycle through $1000^3 = 1{,}000{,}000{,}000$ combinations to check all of the possibilities.

Token Set Size (Continued)

An even larger set size is the Oxford English Dictionary with about 170,000 words. In this case, a three tokens add up to $170,000^3 = 4,913,000,000,000,000$ combinations to check.

XKCD is **Wrong**. "CorrectHorseBatteryStaple" is long, but it is only *four* tokens.

AND

Nobody will use the full set from the OED. They will probably choose from the Thing Explainer Dictionary. So the search space will actually be $1000^4 = 1,000,000,000,000$ combinations to check.

A list of 1,000 words only has about 6.6 bits of entropy for word, not 16 bits of entropy that XKCD claims.

If there is a system, then there is a weakness.

Bootable USB Sticks

Creating a bootable DFSee flash drive has been a major fail. I tried using the bootable CD-ROM with no luck at all. I got a little bit further along with the Linux command versions.

I tried to follow the directions *really, really, really, really* carefully with no luck.

The instructions about creating a bootable USB key say:

> *This can be done on OS/2 or eCS (tested) and on Windows or Linux (both untested, but procedure should be exactly the same)*

Oh, and at the start of the instructions there is also this bit of information:

> *It offers booting on most modern PCs using a 128MB or larger memory stick, some Linux magic, the DFSee bootable CD ISO and a copy of the DFSee linux program.*

I went on a hunt for **Linux magic.**

A Cross Platform Tool for Bootable USBs

I found UNetbootin as a tool to make bootable USB keys. It works on the Mac and I was able to make bootable USB keys for various Linux distributions.

I made a bootable USB key for the Ultimate Boot CD 5.6.3. Wow! All kinds of nifty bootable tools.

AND

Instructions on how to add another ISO to the USB. Instructions that work.

Then I Found Out

I got to talking at the local Hacker Space and had a few recommendations for Windows tools:

- RUFUS
- YUMI

When I tried to put one Linux distribution on USB Rufus told me that it did not have a proper syslinux version for that ISO. Could Rufus download a compatible version.

I approved the download and it worked.

Another Revelation

I then tried another Linux ISO and then Rufus told me there were TWO ways to make a bootable USB. The ISO was a Hybrid/ISO and I could

- Use syslinux and have a FAT32 partition that I can add another Linux ISO to later, or
- Make a direct copy and the USB would be dedicated to that distribution only.

And Then I Tried DFSEE :(

When I fed RUFUS the DFSEE ISO he griped:

> *This is not a bootable-iso, or it uses a boot or compression method that is unknown.*

So the last time I asked: Is there a standard for making a bootable image?

I should have asked: How *many* ways can you make a bootable ISO and convert that to a bootable USB key?

Or as Rudyard Kipling might say:

> *"There are nine and sixty ways of constructing tribal lays,*
> *"And every single one of them is right!"*

(From `http://www.kiplingsociety.co.uk/poems_neolithic.htm`)

Something More Added After the Meeting

Leo Laport and Steve Gibson have discussed passwords on three recent "Security Now" podcasts (#595, #596, and #595). The podcasts are at:

`https://www.grc.com/securitynow.htm`

Their take on XKCD is pretty much the same as what is above. Their discussion, however, did prompt me to do a bit more research.

Steve Gibson did a good job of describing the math behind determining the information entropy of a password.

Anyway, here is some additional notes on the XKCD examples.

The First XKCD Example: Tr0ub4dor&3

XKCD assigns bits of entropy as follows:

- The base word: 16 bits
- Capitalization: 1 bit
- Substitutions: 3 bits
- Punctuation: 4 bits
- Numbers: 3 bits
- Order: 1 bit

Total entropy for this eleven character password: 28 bits

A Random Eleven Character Password

Now assume we are using a password manager to generate an eleven character gibberish password for us. If we use the letters and numbers

$$[0\text{-}9a\text{-}zA\text{-}Z]$$

we have 62 characters and the information entropy for each character in the set is given by:

$$\log_2(62) = 5.954 \text{ bits per character}$$

The total entropy for the complete eleven character password is

$$11 * \log_2(62) = 65.50 \text{ bits}$$

Note that this exceeds the 44 bits of entropy that XKCD has determined for "CorrectHorseBatteryStaple".

An Extended Character Set

We can increase the information entropy when we note that the ASCII character set has additional printable characters. For the complete set of printable ASCII we have 96 printable characters. If we omit the space, then the information entropy for each character in the set is given by:

$$\log_2(95) = 6.570 \text{ bits per character}$$

The total entropy for the complete eleven character password is

$$11 * \log_2(95) = 72.27 \text{ bits}$$

How Much Gibberish?

XKCD gets 44 bits of entropy from a 28 character long, four word pass phrase. We can compare this to the number of characters in a gibberish password needed for 44 bits of entropy. This is calculated assuming letters and numbers to give:

$$\frac{44}{\log_2(62)} = 7.390$$

So a gibberish password with eight characters has the same amount of information as the 28 character pass phrase presented by XKCD. An attack cycling through all of the eight character combinations will take just as long as the four word pass phrase. (This assumes that we have the same dictionary of 2048 common words used by the XKCD system. Note: $2^{11} = 2048$.)

The Trade Offs

The big trade off is that some passwords and pass phrases must be easy to remember. A random 8 character gibberish password is not going to work very well to open the password manager on your phone. In that case, a pass phrase may be suitable.

The second example from XKCD using common words illustrates a pass phrase. However, XKCD does not explain how to make a "good" pass phrase. The Diceware method is one way to generate a random pass phrase. The method is described at:

`http://world.std.com/~reinhold/diceware.html`

Alternate word lists for Diceware are available from the Free Software Foundation at:

`https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases`

The Trade Offs – Continued

Also, some systems may have arbitrary limitations. I have encountered some on line systems that restrict password lengths to 16, 20, or 32 characters. Those systems will choke on "CorrectHorseBatteryStaple" since it is too long. Since the average length of an English word is about 5 characters†, a four word pass phrase may be too long in some cases.

Fortunately, many encryption tools such as GPG place no limits on the length of a pass phrase.

Finally, some systems may insist on special characters. However, the choice of acceptable punctuation may not include the full 95 printable ASCII set. In other words, your mileage may vary.

† V.V. Bochkarevm, A.V. Shevlyakova, and V.D. Solovyev. "Average Word Length Dynamics as Indicator of Cultural Changes in Society," preprint 1208.6109 available from `https://arxiv.org`