Choose the right password

Passwords control all aspects of our lives, but how should we select them? **Julian Bucknall** explains

Passwords seem to be the modern version of the medieval hairshirt. They seem to exist as an irritant to today's online life. You want access to your PC? Password, please. You want to add a Facebook status? Password! You want to check your bank account online? Password needed! So, how do you create good ones? In fact, what are good ones? How do you remember them? How can you reduce the irritation?

In order to authenticate yourself to the systems you use every day – to prove to them that you are who you say you are – you use a password. This password, in theory anyway, is known only to yourself and the system you are trying to access – be it Facebook, Twitter, your bank, your email, your blog or anything else. It is a secret not to be revealed to third parties.

There is another essential piece to the authentication puzzle – your username – but this is generally your email address or your name in some concatenated form, and is easily discoverable. Your password is therefore the 'open sesame' that reveals everything about you. How can you make sure that your privacy remains intact and that the secret persists?

Let's approach the question from the viewpoint of a black hat hacker who wants to impersonate you for some system. To raise the stakes, let's assume that the system is your bank and the hacker wants to test your credit limit. How can he get your password?

Watch and learn

The first way is the simplest: he watches you as you type in your password. That way it doesn't matter how strong or weak your password is; the hacker just watches you enter it. I'm going to assume that you'd be aware of someone watching over your shoulder, so the question becomes how else could a hacker 'watch' you?

Back in March, RSA (producer of the SecurID systems used by corporations and the US Department of Defense) was hacked. Someone managed to gain access to internal systems and networks and steal secrets pertaining to the SecurID two-factor authentication key. A couple of months later, they attempted to hack into Lockheed Martin, the defence contractor using them. How was this done? Simple – it was a phishing attack. An email purporting to be about 2011 recruitment plans and containing an Excel spreadsheet was sent to several low-profile staff members at RSA, seemingly from a recruitment agency. The spreadsheet contained an embedded Adobe Flash object that in turn contained a zero-day vulnerability. Once the spreadsheet was opened, this malware installed a backdoor onto the machine, which gave the attackers access to the PC and the network.

Secret questions

If you forget a password, many websites will ask you a secret question and expect to receive a preset answer before they send you a replacement password. Or the secret question is used in a two-factor authentication process (one of the banks I use employs this scheme).

The problem is that lots of sites use the same set of questions. Not only your mother's maiden name, but also things like the name of your first pet, the street you lived in when you started school, or your favourite movie. The well-read hacker might know many of the answers from reading your social site of preference.

I've solved this by making it up. I've written down a life story of someone else that answers the usual 'secret' questions. The more outlandish the person, the easier it is to remember. For example, for the Queen the answers might be Windsor, corgi, Windsor Castle and The King's Speech. ■

At that point all bets are off. The attacker could install a keylogger and track exactly what you type at login screens – there goes a password. Even worse, they could download your system password files (those used by System Account Manager) and then crack them with a program like Ophcrack, which uses techniques like rainbow tables to reverse the hashed login data. There go *all* your passwords.

In fact, that last scenario brings up the whole subject of cracking passwords. There are two stages: guessing the password using some algorithm – usually brute-force by trying every permutation – and then validating the password against the system being hacked.

The issue with validating passwords is that many systems have built-in safeguards. Generally you only get so many attempts at trying a password before the system locks out the account being tried. Sometimes the system will also deliberately delay resetting the login screen by a few seconds to make trying many passwords extremely slow. Note that a standalone Windows 7 machine has account lockout disabled by default, whereas a PC on a corporate network might have it enabled.

If the system is embodied in a file – say the victim is using a password manager and the hacker has managed to capture the password file – the hacker's job is made much easier. In essence, the online safeguards (limited number of password attempts, delay between attempts) are no longer in play and the hacker has free rein to try as many passwords as they like as quickly as possible. This is where the strength of the password comes into play.

Strength in numbers

When we access a new resource for which we have to create a password, we're generally given some guidelines for creating a strong ▸

password and discouraged from using weak ones. The guidelines usually include making passwords longer than some defined minimum (say, eight characters), not using normal words, using upper and lower case letters, and using numbers and punctuation symbols.

With luck, the screen where you enter your new password will have some kind of visual cue to show how good it is, like a progress bar coloured from red (bad) to green (good).

The worst systems are those that limit your password to a low character count, restrict the characters used to just lowercase letters and digits, and so on. Such guidelines will automatically produce weak passwords.

The strength of a password is measured by its entropy, as a number of bits. The greater the number of bits the larger the entropy, and the harder it will be to crack the password.

Entropy is a concept from information theory, and is a measure of a message's predictability. For example, a series of tosses from a fair coin is unpredictable (we can't say what's coming next) and so has maximum entropy. Text in English – this article, for example – is fairly predictable in that we can make judgments about what's going to come next. The letter E appears far more often than Q, if there is a Q, it's likely that the next character will be U, and so on. It's estimated that English text has an entropy of between one and 1.5 bits per (8-bit) character.

In another sense, entropy is a measurement of how compressible a message is – how much fluff we can discard in compressing a message and still be able to reconstitute the original message at a moment's notice. If you like, the compressed message contains just the information content of the message. We've all compressed a text file in a zip file to get 70-80 per cent compression or more; that is just an expression of the entropy of the text.

Password entropy

Let's apply this to a password. Suppose we are only allowed to use numeric digits in our password. In other words, our password is a PIN that we use to get cash from an ATM. Each character is selected from a set of 10, from 0 to 9. How many bits of entropy are there per character, assuming that each character is going to be selected randomly? First of all, there are eight bits per character using an ASCII character set, but most of those bits can be discarded without losing the 'essence' of the digit. We can compress the characters to a simple binary code: 0000 for 0, 0001 for 1, all the way to 1001 for 9. We can say there are between three and four bits of entropy for each digit (only 8 and 9 need four bits – the rest of the digits need three) and use a bit of mathematics to basically calculate log2(10), which gives us 3.3 bits per digit.

If the digits in the password are chosen randomly (so that the PIN isn't 1111 or 1234, for example), the digits are independent from each other. In other words, knowing one or more digits in the PIN doesn't help us guess the remaining ones. The total entropy in a four-character PIN is about 13 bits.

This means that guessing a four-digit PIN is equivalent to tossing a fair coin 13 times to get a particular sequence of heads and tails. Since there are 2^13 (8,192) different ways to toss a fair coin 13 times, we have some appreciation of how many trials a hacker would have to make in order to break a PIN.

I know there are 9,999 possible different PINs. I've rounded the total entropy down, but

Spotlight on… Password generation techniques

Since memorisation of completely random passwords is so hard, even if they are the best to have security-wise, various techniques have evolved to give us passwords that appear to be random. Note though that these passwords will have reduced entropy compared to fully random ones. Go for slightly longer ones to increase security perhaps.

First point: always avoid using any personal information such as pet names, birthdays, favorite bands and the like. They're all too easily found. You should also avoid normal words obfuscated by adding a few digits on the end. These are all too easily discovered.

Acronyms can produce some strong passwords. Think of a phrase that's meaningful to you with about 10 words in it and use the initial letters as your password. For example, TSoFbSACD could be derived from 'The Sign of Four by Sir Arthur Conan Doyle'.

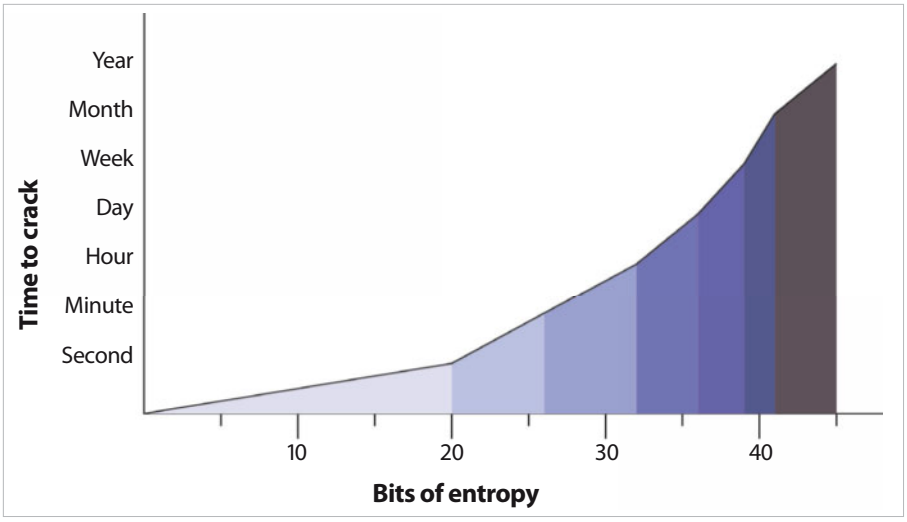
I've used mathematical expressions before, with words replacing parts of the expression. They're nice because it's easy to include digits and punctuation, and there are just so many to invent. Examples are twice7=14, 6!is7twenty, pi=22/7approx, and so on. Aiming for 10 or more characters is probably a good rule. Another technique is to use leet-speak (or 1337-speak) where certain letters are replaced by the digits and punctuation that resemble them. For example, Ju1i4n is my name transcribed to leet. The advice here is to not use a single word (many rainbow systems will encode the standard leet-speak versions), but concatenate at least a couple. You can try just modifying some characters, leaving others alone, but it gets hard to remember which. Once you seed it with punctuation at the start or end, it gets harder still.

By far the best option is to use a password manager religiously and get it to generate strong random passwords that you can copy and paste when needed. All you then need is one super-strong password to access the program. All these techniques are great for that, and the longer it is the better. ■

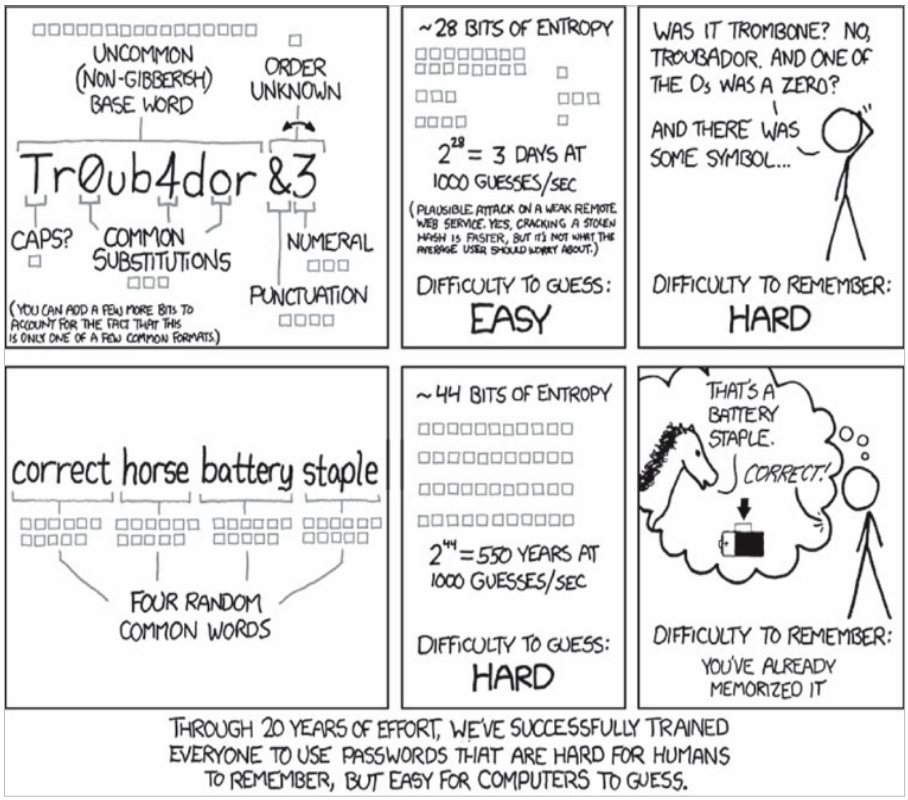
Special characters

Some systems allow the use of special characters like accented characters, Greek letters or the German double-S. I used to use these for some of my passwords, but I found myself running into a couple of problems. The first is the assumption that you will never use another country's keyboard to enter a password. If you've used a special character, you may find it difficult to generate it on a keyboard with a different layout. Even some standard punctuation may not be available on another country's keyboard.

Another problem arises when you have to type the password on a mobile device. If you do use a special character or two, learn how to generate them on your smartphone or you may lock yourself out of your password-protected resource. It is for this reason I stick to standard punctuation (as well as alphanumeric characters) for my passwords. ■



▲ **Figure 1: Potential increase in cracking time with linear increase in entropy**



▲ **This cartoon from XKCD says it all – our usual passwords are very easy for machines to break.**

the error is insignificant and using bits of entropy makes the estimates for cracking a password easier to understand. Bear with me.

Now let's look at it from the hacker's viewpoint again. Let's say that using some specialised password-cracking programs, a hacker might be able to generate and try one million passwords per second. One million is roughly 2^20, so another way of looking at this is that our hacker can test 20 bits of entropy per second. Our PIN number would fall instantly. Luckily the issue with hacking PINs is the validation of them: hopefully your bank would lock the account after three invalid attempts or so. Still, this is a nice round number for evaluating the strength of a password: a password with an entropy of 20 bits will be cracked in one second.

Also, since there are approximately 2^25 seconds in a year, we can estimate that our virtual hacker will crack a password with an entropy of 45 bits in a year. We'll call such a password a year-strong password.

Since every extra bit of entropy doubles the cracking time, we can estimate that a 50-bit password will take 32 years to crack. Doubling the speed of cracking will halve the time taken, and therefore require an extra bit of entropy to get us back to where we were.

Character traits

Now that we have a feel for the strength of passwords using entropy, we can try using different character sets for our passwords. For now we'll assume that each character in a password is chosen randomly; we'll talk about what happens if this is not the case later.

Web passwords

We've all been there: we visit lots of websites while browsing the web and we're prompted to enter a password for each one. How can we generate enough passwords that we can remember? Using the same one for everything is frowned upon: if one website gets hacked, the hackers have your access to everything.

One technique is to create a single strong master password (say, for sake of argument, 6!is720) and then modify it for each website. One method is to insert the name of the site in question into the password, say at the third position, so we'd get 6!facebookis720, 6!twitteris720, 6!amazonis720, and the like.

Since that involves a lot of typing, another method is to encode the website name somehow and insert the encoding into the password. For example, let's encode the website name as the initial letter, capitalised, followed by the number of letters in the site's name. We'd have 6!F8is720, 6!T7is720, 6!A6is720 for the sites previously discussed. You can use the last letter, or enter 10 minus the number of letters, use a different position, or any other encoding you want. ■

Let's add the characters A to F to our set of possible symbols. This is what WEP passwords were like on your old Wi-Fi router (WEP was deprecated in 2004). There are exactly four bits of entropy per character. A 10-character WEP key (the original standard) would have 40 bits of entropy. A brute force attack would discover it in 2^20 seconds, or 11 days. WEP suffers from other security issues, so a brute force attack wouldn't be needed in practice.

Now let's look at just using single case letters to form a password. Since there are 26 of them, we have 4.7 bits of entropy per character (2^4.7 = 26). Let's suppose we want to have a year-strong password, then we would have to have a 10 letter password, with each letter being completely random.

If you're using uppercase, lowercase and digits, that's a 62 element set, or just under six bits per character. A year-strong password would need eight characters, and these would need to be completely random.

Adding punctuation like commas, semicolons, question marks and so on would give us another 16 possible characters, to make 6.3 bits of entropy per character. A year-strong password would need about seven characters.

The biggest problem for us as humans when presented with completely random passwords is memorising them. It's possible with one eight-letter random password I suppose, although I'd hate to, but several of them would be a chore, especially if they involved punctuation. A better option is to generate quasi-random (or random-looking) passwords as described on p88. You could say these types of passwords have mnemonics built in and are nothing like '123456' or 'password'.

While we're discussing entropy and character sets, let's play around with another type of symbol set: the set of all words. To be more specific, suppose we have a list of 2,000 words. The entropy per word is 11 bits, since 2^11 is roughly 2,000. How many random words from this list concatenated together would produce a year-strong password? The answer is, surprisingly, roughly four. If each word is seven letters long or fewer, you'd be typing in 28 characters or fewer for your password. If the 2,000 words in the list were specially chosen to help evoke images in your mind, memorising the four-word password would be much easier. Unfortunately, few services will allow a 28-character password.

And how would you choose the words randomly? A computer program is one way, but if you just have the numbered list of words, you could try shuffling a pack of cards. Take out the court cards. Shuffle the rest well and deal out three. Counting 10 as zero and ignoring suits, you can read off a four-digit number between 0 and 999. Now check the colours shown: if you have more reds than blacks, add 1,000 to your number. You now have a random number referencing one of your words in the list. Repeat this three more times to get the four random words.

As a final word, let's repeat the winner of the Best Gag award at the 2011 Edinburgh Fringe Festival. It was by Nick Helm and went as follows: "I needed a password eight characters long, so I picked Snow White and the Seven Dwarves." And on that note, I'm logging off and changing my password. PCP

Julian M Bucknall has worked for companies ranging from TurboPower to Microsoft and is now CTO for Developer Express. *Thanks to GData for its support in creating this article.*
feedback@pcplus.co.uk