Generating good passwords

Password security is very much on everyone's mind. Either you are a user, who gets irritated because of the funky passwords some web sites make you use, or you are a systems administrator, frustrated that your users want something simple, putting the systems you manage at risk of being compromised.

Summary

If you don't want all the ins and outs, just do one of the following. For the reasons why you should do one of them, read the rest of the article. If you want to jump straight to the complex explanation, read password_theory and secure_memorable_passwords

Generate a totally random, 20 character password

This is the hardest to guess, and the hardest to remember. You have to write it down, and, more importantly, you have to type it in. It should contain every character you can type on a computer keyboard, so you end up with something like 'm“#;{k[R.<_Ta}5W2zgZ' (generated from https://passwordsgenerator.net/). Some online accounts will not let you use this, it is difficult to remember, and you have a very, very good chance of messing up typing it.

Use five, randomly chosen dictionary words, separated by a special character

This sounds completely counter-intuitive, but it is probably the best way to generate passwords. With this, you end up with a password like 'high.ANYTHING.also.NECK.girl' (generated from https://xkpasswd.net/s/), which is fairly easy to type, fairly easy to remember, and works with most online accounts. You may have to add a numeric digit for some web sites, so 'high.ANYTHING.also.NECK.girl.45' or something.

Pad your existing password

This is not as good as either of the above, and stands a much better chance of compromise if there is any social engineering going on, but simply adding characters to an existing password to make it longer will make it more secure. Let's say I have a password I have used forever, and I have it memorized, and I am madly in love with it. It used to be a good password, 20 years ago, but this obscure dictionary word is totally useless now. I'll make up one, based on the name 'Tiffany'. I've used 'T1ff@ny' for a long, long time, but as you can see further on, it is not a good password anymore.

What I do is add one or two single characters repeatedly in front and back to make it 20 characters long. I'm going to go crazy here and put 86 in front and back, continuously, until it is 20 characters long. So, I end up with '86868686T1ff@ny868686' (I added 5 repetitions in front, 4 at the end, and got 21 characters). As long as no one knows how you got there (it is the year of her birth), this is a very, very difficult password to guess. Even if the T1ff@ny password is known, it is still hard to guess. Not

as good as the five, randomly chosen words above, but definitely better than just adding a number to the end, as we have a tendency to do.

I want to know more

Ok that is it. If you're willing to just take my word for it, and don't care about why, end of the article. However, for the curious amongst you, the rest of the this document explains why.

There are three basic ways the Black Hats (someone who is trying to “get” your password) work. In order of difficulty (easiest to hardest), they are

Compromise a server

The Black Hat's crack a server and steal the encrypted passwords stored on it. This is one of the main targets of penetration attempts (attempts to “get into” someone's server), and they are going on all the time. For a list of recent successful attacks that have been reported, I usually look at https://haveibeenpwned.com/

If successful, the black hats can use a lot of tricks to try to find out what password you used. Getting one of these lists is the most profitable source for them, since they can get hundreds of thousands, or even hundreds of millions, passwords to try, and usually additional information like your e-mail account, name, etc… One successful penetration can result in millions of known passwords.

Most people use the same passwords, or slight variations of them, everyplace they go. So, once Black Hats know one of your passwords, say from your Utility Company, they have a good chance of knowing your bank, social media and e-mail passwords.

Social Engineering

They learn about you. What does Facebook say about you. Do you like dogs? What is your birthday. What is your spouse/significant other's name. From this, they can guess what kind of passwords you might come up with. Try passwords that are composed of breeds of dogs, or your spouses name.

Combined with a compromised server, this can give them access to your other accounts with minimal effort. They now know what kind of passwords you use, so if you've only used a variation of it some place else, it is easier to figure out more difficult passwords.

Brute Force

This is the least profitable, but the easiest to do. They simply have a bunch of computers try to continuously log into your e-mail account, or your blog. However, if they compromise a server, then do social engineering to learn more about you, they can then use Brute Force to try several variations on your other sites. On our mail servers, we see continuous attempts at this kind of attack, even though we have automated systems that detect, then block them. When we block the server it is being tried from, a few hours (or minutes) later, it resumes from a new machine.

What do you do

First, use different passwords as much as possible. You've heard “don't write passwords down,” which is wrong. In a perfect world, with perfect memory, everyone would have a completely different password for everything, and be able to remember them all. However, for the rest of us, it is better to have a list someplace safe. A small piece of paper in your wallet, or in a locked desk drawer is much better than using the same single word password every place you go. You can have a favorite color, but not a favorite password. Just make sure you secure the list.

Security expert Bruce Schneier, one of the most respected names in in the field, recommends this approach. See https://www.schneier.com/blog/archives/2005/06/write_down_your.html. Secure it like you do your driving license, or credit cards, and if it is stolen, change all the passwords at the same time you are replacing those documents.

However, since Dr. Schneier wrote this article, several applications for computers and cell phones have been created that can store your passwords securely. That is better than a piece of paper, but I still like the paper copy (stored in a safe), in addition to these apps.

How to choose a password

Good passwords have the following characteristics:

- They are in no way associated with you, the person.
- They are of sufficient length
- They can possibly use as many characters as possible

Randomness

First, as we mentioned in Social Engineering, a password should be completely random, to remove any association with you that social engineering could find. Bottom line is, your password should be nothing YOU would come up with. It needs to be randomly generated (but we can make it easy to use anyway, read on).

Don't use your kitty cat's name as part of the password, or your spouse, or your parents, or your offspring. No dates. Better to open a dictionary and randomly choose several words (more on that later).

Length

Longer is better. For every character you add, your strength grow exponentially. An 8 character, randomly generated password using all of the characters on a keyboard can be guessed in about a minute with a botnet. Adding just one character changes that to 2 hours. Making it 12 characters long increases to 2,000 years. The following table shows the differences.

Length	Time it takes on a medium botnet
8	1 minute
9	2 hours

Length	Time it takes on a medium botnet
10	6 days
11	2 years
12	2000 years
20	11 quintillion years

That last number is large. An 11, followed by 18 zeros. Why would we ever think to do that? Mainly, for the future. In 1999, it would have taken almost 3 years to crack the password 'T1ff@ny' by available technology. In 2010, it would have been cracked in 2 months, 1 week. Today (October 2020), it could be cracked in about a second using a medium sized botnet. Computer systems increase in power over time, and by 2030, that huge number could be down to a few years.

Name Space

Finally, it should use as many different possible “entities” (think characters on the keyboard) as possible. An 8 character password composed strictly of lower case letters is 16 times easier to crack than if you simply add a numeric digit or two. The following list shows the amount of time it takes a standard desktop computer to crack an 8 character password (they do not use standard computers).

Time to Crack	Namespace
35 minutes	all lower case
8 hours	all lower case and numeric digits
6 days	Lower and Upper case
25 days	Lower, Upper, numeric digits
2 years	lower, upper, numeric, special characters (period, comma, colon, hash mark)

I used the Desktop Computer to show the increase in difficulty. Most Black Hats are using special equipment, called GPU's, or clusters of computers (called botnets) to do the cracking. An 8 character password with lower, upper, numeric and special characters can be cracked on a medium sized botnet in about a minute, instead of 2 years.

Other tricks, and how to avoid them

Password duplication

If one of your accounts is successfully compromised, it is assumed you will use the same, or similar passwords on other sites. The average user maintains 25 separate online accounts, but just uses 6.5 passwords on average (https://www.microsoft.com/en-us/research/publication/a-large-scale-study-of-web-password-habits/?from=https%3A%2F%2Fresearch.microsoft.com%2Fpubs%2F74164%2Fwww2007.pdf)

Dictionary Attacks

Passwords are not stored “in the clear” (ie, the original password in text form) on reputable sites. Instead, they use a mathematical function called a hash to turn your password into a big number. The

good thing about the hash is, it is very easy to calculate the hash of a password, but very difficult to calculate the password when given the hash. In other words, it is not easily reversable. When you log into a web site and provide your password, the web site calculates the hash of what you entered, then compares it the the result they have stored, and if they match, you're allowed access. When the Black Hats “steal your password”, the are actually stealing the hash of your password in most cases. They then have to go through the long process of trying to reverse it, which can take years (in some cases, millions of years).

Instead of trying the very time consuming process of reversing the process for every hash they steal, they have calculated list of hashes for known or suspected passwords. Then, all they have to do is look to see if that hash has already been pre-calculated and, if so, they know your password. It took a long time, but there are lists available that have the hashes for every word in the dictionary (in almost all languages). In addition, there are lists of hashes for passwords that have been stolen before, and passwords that have been created by modifying the other lists slightly. To see if a password is in any of the known lists (with emphasis on the word 'known' as there may be more that are hidden), visit https://haveibeenpwned.com/Passwords or http://unixservertech.com/pwned/pwned.html. This is a simple script that checks for known passwords without (I repeat, without) sending the password. We just send part of the hash (see explaination at http://unixservertech.com/pwned/pwned.html, or details by the author at https://www.troyhunt.com/ive-just-launched-pwned-passwords-version-2/#cloudflareprivacyandkanonymity). If one of those sites (and they use the same back end, so no reason to check both) tells you your password is known, it is trivial for someone to learn your password if they find the hash in someplace they have attacked.

Brute Force

If the hash is not in one of the pre-calculated lists, it is more difficult to figure out. Previously, we mentioned that it would take 2 years using a standard desktop computer to reverse calculate a password that was 8 characters long. However, realize, the Black Hats are not using Standard Desktop Computers!

They will use a special computer with something called a GPU (Graphics Processing Unit). This device, designed to make your computer screen show complex changes rapidly, can also be abused to crack passwords. Some of them will have several GPU's in their computers, working together. Others will use special botnets, where hundreds, or even thousands of computers attack the same problem at the same time.

All of these options shorten the time it takes to crack a hash and figure out a password. Let's take a simple example. The password 'af2r@bcG!' is 9 characters long and uses all of the keyboard. It would take about 2000 years to crack on a standard desktop PC. But, what happens when more advanced hardware, or networks, gets into the act and uses brute force to crack it. This information was found by putting the password into the checker at http://password-checker.online-domain-tools.com/

Time	Equipment
2,000 years	Standard Desktop PC
46 years	Fast Desktop Workstation
18 years	Workstation with a single GPU
9 years	Workstation with a single, fast GPU
11 month	Workstation with parallel GPU's

Time	Equipment
2 hours	medium size botnet

So, as you can see, 2 hours after someone running a botnet puts it to work on your password, they will know it. And, store the hash, and the known password, into the lists freely available on the Internet.

Password Theory

There are two things that affect how long it takes to crack well formed (randomly generated) passwords; the speed with which the hardware/software can attack it, and the average number of tries it will take to do so. Speed of processing is increasing exponentially over time, and we have no control over that. The complexity, however, we can control.

Password cracking via brute force is determined by the number of possible combinations you could have. This is calculated as namespace raised to the length power, or namespace^length. Namespace is the number of possible elements in the key. If you only use lower case letters, it is 26 for English. If you use lower and upper case, it is 52. Add in numbers and it is 62. All ASCII printable characters (what you can enter from the keyboard) is 95.

Namespace	Length			
26	8	26	**8, or 208,827,064,576**	lower case alphabetics
52	8	52	**8, or a 9 followed by 13 zeros**	Add upper case letters
62	8	62	**8, or a 2 followed by 14 zeros**	Add numbers
95	8	95	**8, or a 6 followed by 15 zeros**	Everything on the keyboard

Those numbers are very large, so we take the log2 (log base 2) of them and call that the entropy of the password. In the above case, the entry would be 37.60, 45.60, 47.63 and 52.56 respectively, Much easier to write down and remember. The formula for calculating the entropy

entropy = log2(namespace ^ length)

On average, you will guess the correct password when you are half way through the list of possibles. If you have 1000 things to check, you should, on average, be done after checking 500 of them. Some will take longer, some will be shorter. To divide by the possible combinations by 2, using entropy above, simply subtract 1 from the entropy (it is a log). Therefor, the number of guesses to crack a password, on average, will be

```
number_of_guesses = 2^(entropy-1)
```

And the average time to guess a password, in seconds, would be

```
time_to_guess = 2^(entropy-1) / guesses_per_second
```

Secure, Memorable Passwords

Let's look at two variants to measure the strength of a password: If the attacker does or does not

know how you created it. If the password is totally random, using all printable characters, the entropy is the same for both scenarios.

Knowing the length of the password decreases the entropy for smaller and smaller lengths. For example, the entropy of a 8 character passphrase of all printable characters is 52.5. However, if the attacker does not know the length, they must try everything commonly used, so the entropy would be 78 for, say a length of 12. In reality, they would probably check all the 8 character's first, so they would get a hit much faster, but the knowledge vs lack of knowledge effects are easier to see here.

With passwords created in either of the following two ways, the entropy will vary widely with knowledge. Note: I have read older documentation where it is suggested to keep your entropy above 52 bits. However, I think we can hit a trillion (1E12) guesses per second over the next 10 years, which would reduce 52 bits to cracking in about a half hour (38 minutes). I prefer an entropy of about 100 which, if my calculations are correct, would take at least a century to crack.

Haystack (aka Padding)

I found a very interesting article about “haystacking,” a process which takes a poor password and turns it into a very strong one. See https://www.grc.com/haystack.htm.

The process is simple. Take a poor password, say 'T1ff@ny', then pad it with a single (or short sequence) of other characters, to make it larger. As described near the beginning of the article, we take that, then pad it to make it a 20 character password. The “blind” entropy of this (they don't know how we did it) is 131, which is good. The author makes a point that, even if they know the password is haystacked, it is still good. I agree with one caveat. If they can use social engineering (or previously collected old passwords of yours), they can greatly reduce the search space. I have not seen any research into this concept, but it sounds good. Kinda. Sorta.

However, I'd be leery of doing this with anything like my bank or something if there is a chance that some knowledge may be available to the attacker. If they know you have an old password that you liked, and you were using haystacking, the number of possiblities could be reduced into the thousands and be cracked faster than it would take to enter the command to do it.

Diceware (random word list)

This is a weird one, but definitely proven. It creates a passphrase composed of randomly selected words, separated by a special character. https://xkpasswd.net/s/ in the XKCD preset, is a tool to create these, but I recommend a minimum of 5 words as that increases the “full knowledge” entropy to over 100. For the technically oriented amongst you, you can download the source code driving that site from https://github.com/bbusschots/hsxkpasswd and create your own dictionary for very good security.

This is an example Diceware. The procedure is fairly simple, though tedious, and can be done by hand.

1. Create a list of 7,776 words from the dictionary
2. Roll 5 die, using the result to look up a single word in the list
3. Repeat until you have found 5 words
4. Put the words, in order, into a phrase, separating them by a special character

This will result in something like 'mexico-bread-inside-this-factors', which is good, but by randomly making words capitalized, could end up being 'MEXICO-bread-inside-THIS-FACTORS', which is not much harder to remember, but much more secure (raises entropy by one). And, the human mind can turn this into something memorable.

The site above uses a known dictionary (the Black Hats know it), and even with that, the entropy on this is 102 bits, even if they know how you did it. You can enhance it even more by generating your own word list, in which case it becomes even more difficult.

Theory

The wikipedia article at https://en.wikipedia.org/wiki/Diceware explains this well, but I'll summarize. The number of possible combinations is namespace ^ length, from above. Increasing length definitely has a strong effect, but increasing namespace has a greater one (2^2 = 4, 2^3 = 8, however 3^2 = 9). In this case, we are increasing the namespace from 95 (number of ASCII characters) to 7776 (number of words in the list), or 15552 (if we randomly capitalize). Ignoring the separators, that gives us a huge namespace increase, meaning we don't need to have as many “tokens” (characters under standard passwords vs words with Diceware). 5 randomly chosen words from the word list gives us a whopping 9E20 (9 followed by 20 zeros) possible combinations, or an entropy of 69 bits. Adding in the special characters you can put in between, and you get an entropy of over 100.

If your list only includes words between 4 and 8 characters, the “blind” entropy is even greater, running between 150 and 275 bits (the key length will be between 20 and 40 characters, with an average of 30).

All this in an easy to remember, easy to write down, easy to type in, passphrase.

Links

- http://password-checker.online-domain-tools.com/
- https://generatepasswords.org/
- https://www.grc.com/haystack.htm
- https://en.wikipedia.org/wiki/Password_strength#Entropy_as_a_measure_of_password_strength
- https://resources.infosecinstitute.com/password-cracking-evolution/
- https://en.wikipedia.org/wiki/Diceware
- https://www.microsoft.com/en-us/research/publication/a-large-scale-study-of-web-password-habits/
- https://xkpasswd.net/s/
- https://github.com/bbusschots/hsxkpasswd

== Perl script to calculate stats on a way to generate passwords

This is just something I threw together. No comments, half the code commented out. Feel free to use it however you want, but don't sue me if I made a mistake and you get pwned because the script told you something was good.

#! /usr/bin/env perl

use strict; use warnings;

my $calculationsPerSecond = 1e12;

sub log2 {

```
return log( shift ) / log(2);
```

}

sub secondsToYD {

```
my $seconds = shift;
my $secondsPerDay = 86400;
my $secondsPerYear = $secondsPerDay*365.2425;
my @return;
```

```
# seconds
```

unshift @return, sprintf(”%02d“,$seconds % 60); # $seconds = int($seconds / 60);

```
# minutes
```

unshift @return, sprintf(”%02d“,$seconds % 60); # $seconds = int($seconds / 60);

```
# hours
```

unshift @return, , sprintf(”%02d“,$seconds % 24); # $seconds = int($seconds / 24);

```
# days
```

unshift @return, sprintf(”%03d“,$seconds%365.2425); # unshift @return, int($seconds/365.2425);

return join(':', @return);

```
return int($seconds / $secondsPerYear);
```

}

my $length = 20; my $namespace = 95; #7776*2; my $combinations = $namespace ** $length; my $entropy = log2($namespace ** $length); my $guesses = 2 ** (log2($namespace ** $length)-1); my $years = (2 ** (log2($namespace ** $length))) / $calculationsPerSecond ;

print “$length\tNumber of symbols\n”; print “$namespace\tUnique symbols available\n”; print “$combinations\tNumber of combinations available ($namespace \^ $length)\n”; print “$entropy\tEntropy (ln($namespace \^ $length))\n”; print “$guesses\tNumber of guesses for 50%\n”; print “$years\tNumber of seconds to guess\n”;

print &secondsToYD($years) . “\tYears to guess\n”;

#print “For 121 bits, ” . 2**121 . ” seconds or “ . &secondsToYD(2**121) . “\n”;