Choosing Secure Passwords

Ever since I wrote about the 34,000 MySpace passwords I analyzed, people have been asking how to choose secure passwords.

My piece aside, there's been a lot written on this topic over the years -- both serious and humorous -- but most of it seems to be based on anecdotal suggestions rather than actual analytic evidence. What follows is some serious advice.

The attack I'm evaluating against is an offline password-guessing attack. This attack assumes that the attacker either has a copy of your encrypted document, or a server's encrypted password file, and can try passwords as fast as he can. There are instances where this attack doesn't make sense. ATM cards, for example, are secure even though they only have a four-digit PIN, because you can't do offline password guessing. And the police are more likely to get a warrant for your Hotmail account than to bother trying to crack your e-mail password. Your encryption program's key-escrow system is almost certainly more vulnerable than your password, as is any "secret question" you've set up in case you forget your password.

Offline password guessers have gotten both fast and smart. AccessData sells Password Recovery Toolkit, or PRTK. Depending on the software it's attacking, PRTK can test up to hundreds of thousands of passwords per second, and it tests more common passwords sooner than obscure ones.

So the security of your password depends on two things: any details of the software that slow down password guessing, and in what order programs like PRTK guess different passwords.

Some software includes routines deliberately designed to slow down password guessing. Good encryption software doesn't use your password as the encryption key; there's a process that converts your password into the encryption key. And the software can make this process as slow as it wants.

The results are all over the map. Microsoft Office, for example, has a simple password-to-key conversion, so PRTK can test 350,000 Microsoft Word passwords per second on a 3-GHz Pentium 4, which is a reasonably current benchmark computer. WinZip used to be even worse -- well over a million guesses per second for version 7.0 -- but with version 9.0, the cryptosystem's ramp-up function has been substantially increased: PRTK can only test 900 passwords per second. PGP also makes things deliberately hard for programs like PRTK, also only allowing about 900 guesses per second.

When attacking programs with deliberately slow ramp-ups, it's important to make every guess count. A simple six-character lowercase exhaustive character attack, "aaaaaa" through "zzzzzz," has more than 308 million combinations. And it's generally unproductive, because the program spends most of its time testing improbable passwords like "pqzrwj."

According to Eric Thompson of AccessData, a typical password consists of a root plus an appendage. A root isn't necessarily a dictionary word, but it's something pronounceable. An appendage is either a suffix (90 percent of the time) or a prefix (10 percent of the time).

So the first attack PRTK performs is to test a dictionary of about 1,000 common passwords, things like "letmein," "password," "123456" and so on. Then it tests them each with about 100 common suffix appendages: "1," "4u," "69," "abc," "!" and so on. Believe it or not, it recovers about 24 percent of all passwords with these 100,000 combinations.

Then, PRTK goes through a series of increasingly complex root dictionaries and appendage dictionaries. The root dictionaries include:

- Common word dictionary: 5,000 entries
- Names dictionary: 10,000 entries
- Comprehensive dictionary: 100,000 entries
- Phonetic pattern dictionary: 1/10,000 of an exhaustive character search

The phonetic pattern dictionary is interesting. It's not really a dictionary; it's a Markov-chain routine that generates pronounceable English-language strings of a given length. For example, PRTK can generate and test a dictionary of very pronounceable six-character strings, or just-barely pronounceable seven-character strings. They're working on generation routines for other languages.

PRTK also runs a four-character-string exhaustive search. It runs the dictionaries with lowercase (the most common), initial uppercase (the second most common), all uppercase and final uppercase. It runs the dictionaries with common substitutions: "$" for "s," "@" for "a," "1" for "l" and so on. Anything that's "leet speak" is included here, like "3" for "e."

The appendage dictionaries include things like:

- All two-digit combinations
- All dates from 1900 to 2006
- All three-digit combinations
- All single symbols
- All single digit, plus single symbol
- All two-symbol combinations

AccessData's secret sauce is the order in which it runs the various root and appendage dictionary combinations. The company's research indicates that the password sweet spot is a seven- to nine-character root plus a common appendage, and that it's much more likely for someone to choose a hard-to-guess root than an uncommon appendage.

Normally, PRTK runs on a network of computers. Password guessing is a trivially distributable task, and it can easily run in the background. A large organization like the Secret Service can easily have hundreds of computers chugging away at someone's password. A company called Tableau is building a specialized FPGA hardware add-on to speed up PRTK for slow programs like PGP and WinZip: roughly a 150- to 300-times performance increase.

How good is all of this? Eric Thompson estimates that with a couple of weeks' to a month's worth of time, his software breaks 55 percent to 65 percent of all passwords. (This depends, of course, very heavily on the application.) Those results are good, but not great.

But that assumes no biographical data. Whenever it can, AccessData collects whatever personal information it can on the subject before beginning. If it can see other passwords, it can make guesses about what types of passwords the subject uses. How big a root is used? What kind of root? Does he put appendages at the end or the beginning? Does he use substitutions? ZIP codes are common appendages, so those go into the file. So do addresses, names from the address book, other passwords and any other personal information. This data ups PRTK's success rate a bit, but more importantly it reduces the time from weeks to days or even hours.

So if you want your password to be hard to guess, you should choose something not on any of the root or appendage lists. You should mix upper and lowercase in the middle of your root. You should add numbers and symbols in the middle of your root, not as common substitutions. Or drop your appendage in the middle of your root. Or use two roots with an appendage in the middle.

Even something lower down on PRTK's dictionary list -- the seven-character phonetic pattern dictionary -- together with an uncommon appendage, is not going to be guessed. Neither is a password made up of the first letters of a sentence, especially if you throw numbers and symbols in the mix. And yes, these passwords are going to be hard to remember, which is why you should use a program like the free and open-source Password Safe to store them all in. (PRTK can test only 900 Password Safe 3.0 passwords per second.)

Even so, none of this might actually matter. AccessData sells another program, Forensic Toolkit, that, among other things, scans a hard drive for every printable character string. It looks in documents, in the Registry, in e-mail, in swap files, in deleted space on the hard drive ... everywhere. And it creates a dictionary from that, and feeds it into PRTK.

And PRTK breaks more than 50 percent of passwords from this dictionary alone.

What's happening is that the Windows operating system's memory management leaves data all over the place in the normal course of operations. You'll type your password into a program, and it gets stored in memory somewhere. Windows swaps the page out to disk, and it becomes the tail end of some file. It gets moved to some far out portion of your hard drive, and there it'll sit forever. Linux and Mac OS aren't any better in this regard.

I should point out that none of this has anything to do with the encryption algorithm or the key length. A weak 40-bit algorithm doesn't make this attack easier, and a strong 256-bit algorithm doesn't make it harder. These attacks simulate the process of the user entering the password into the computer, so the size of the resultant key is never an issue.

For years, I have said that the easiest way to break a cryptographic product is almost never by breaking the algorithm, that almost invariably there is a programming error that allows you to bypass the mathematics and break the product. A similar thing is going on here. The easiest way to guess a password isn't to guess it at all, but to exploit the inherent insecurity in the underlying operating system.

This essay originally appeared on Wired.com.