MIKE HOWARD

how often should you change your password?

Mike Howard came into programming from systems engineering and has been stuck there. He currently makes his living doing custom software and system administration for a few small companies.

mike@clove.com

FOLKLORE TELLS US THAT WE NEED to change our passwords fairly frequently. In fact, it is required by the security policy of many companies. I recently revisited the problem and have concluded that *changing passwords doesn't really matter*.

The issue came up because a client for which I have worked for about twenty years recently sold two of its divisions to a major company. The new owners—as you would expect—have all kinds of security requirements and security specialists and policies and what not. In contrast, our security policy was almost nonexistent, not enforced, not policed, and implemented only on privileged accounts. In spite of this, we have never been successfully penetrated from the outside.

Admittedly, the company is not a significant target, but looking at the *secure* logs on the Linux systems on our public networks revealed that they had been subject to fairly continuous, automated, low-level probing. I once made the mistake of misconfiguring an Apache server so it could become an open mail relay and that was almost instantly discovered and used—so automated probing is effective.

During the time we had public networks—well over eight years—we had only changed the root password once and had had that same administrative password on all the servers. We have had dial-in connections for well over twenty years without a break-in. (Although attack through a dial-in port seems unlikely, I personally learned early on that even an unlisted phone number is not safe: I had an early Xenix system cracked when I sloppily configured it *without* a root password.)

So I got curious.

This note is a summary of what I came up with. A more detailed note, "How often should you change your password—or should you bother?" is available at www.clove.com/clove_tech/tech_notes.

Throughout this note, I use the letter k to denote the cumulative number of password crack attempts, N for the total number of probable passwords, and P_k for the probability of being cracked after k crack attempts.

Effect of Changing Passwords: P_k

If you think about it for a minute, cracking passwords is a classical "drawing with (or without) replacement" probability problem. If I have a sack containing $N - 1$ black balls and a single white one, then guessing a password is equivalent to pulling a ball out of the sack. If I leave the ball out, then that is the same as not changing my password. If I put it back each time, then that is changing my password after each attempt.

In the first case, $P_k = k/N$.

In the second case, $P_k = 1 - (1 - 1/N)^k$.

For anything else, the probability must be someplace in between.

I made the reasonable assumption that we want to keep $k << N$—that is, we want our number of crack attempts to be much less than the number of probable passwords. If this is true, then we get a good approximation for the "continuously changing password" case of $P_k \approx k/(2N)$.

So, in general $k/(2N) \leq P_k \leq k/N$, where you need to take the left $\leq$ with a small dose of salt.

So, no matter how often you change your password, it doesn't have much effect on the P_k—the probability that your password will be cracked.

All that matters is keeping k small relative to N. In other words, we want k (number of crack attempts) small and N (number of probable passwords the cracker has to probe) large.

Probable Number of Password: *N*

The number of probable passwords a cracker has to test is under the control of the users. That is, we usually pick our own passwords. As a result, passwords are generally constructed of words, pseudo-words, and numbers. A few brave souls add punctuation and other things, but my sense of things is that this is rare. For the most part, we use words.

This is important, because our use of words severely limits the size of N. For example, if we construct five-character passwords from the lowercase alphabet alone, we will have 1.2×10^7 possible symbols. Words, however, are more restrictive.

To get a handle on this, I downloaded the *King James Bible*, *The Federalist Papers*, and *A Short Biographical Dictionary of English Literature* from the Gutenberg Project and wrote a little code to do a naive analysis. It turned out that the three texts only contained 10^6 words, of which 2.7×10^4 were distinct. In fact, there were only 2,283 five-character words and only 3,271 eight-character words.

It is clear that using words is dangerous, because N is far too small.

In our case, we generate passwords using a system based on folklore:

- We create a five- to eight-character alphabetic string by selecting a single word or concatenating two short ones together.
- We then randomly capitalize a few letters.
- Finally, we insert a couple of digits.

When I analyzed this scheme using the words in my aforementioned documents, it expanded the number of eight-character passwords from 3×10^3 to 8×10^{12}. If you are willing to use ten-character passwords, this goes up to 7×10^{14}. Again, see "How often should you change your password—or

should you bother?" for more details at htttp://www.clove.com/clove_tech/tech_notes.

The Teracrack project [1] used a dictionary from "crack" containing about $5x10^7$ passwords to create a precomputed password hash of about $2x10^{11}$ entries. Briefly, the Teracrack project tested the feasibility of speeding up cracking by precomputing password hashes used by the crypt algorithm. The project was able to demonstrate that crypt is unsafe because the hash database fit within 1.5 terabytes of disk—an amount easily affordable nowadays. If the size of the password hash scales roughly linearly, then this algorithm should expand the hash requirements by five orders of magnitude—say, to something on the order of 10^{16} for eight-character passwords—and so require something on the order of $1.5x10^5$ terabytes. This is a fairly simple change which should move the storage requirements of this technique out of reach for a little longer. This is *not* an endorsement of crypt, which has been known to be deficient for at least twenty years.

Variations of this scheme yield different effects. Here are some rules of thumb:

- Adding one character of password length increases N by a little less than a power of 10.
- Inserting an additional digit increases N by about a power of 10.
- Replacing letters with digits helps a little, but much less than insertions.

Our experience is that passwords of this form are fairly easy to type from memory—after practicing a few times—even though it is often difficult to remember how to write them down with a pencil.

As a side project, I looked at the effect of case-insensitive password schemes—which are common with certain large computer manufacturers. In general, they reduce the number of probable passwords using this scheme by about two orders of magnitude for short passwords and three for longer passwords (of length 11 or 12).

Case sensitivity in password schemes is very important.

Controlling *k*

The value of k is not static. It grows as crackers attempt to crack and system administrators and system software vendors do nothing about it. It has nothing to do with users and their changing of passwords.

Crackers appear to be building nets of robots that mechanically attack systems. There is not much we can do about that other than detect crack attempts and shut them down.

I made some off-the-cuff computations and came up with an estimate of $3x10^8$ crack attempts per year per host by assuming that the system is exposed to one crack attempt per second. This gives a P_k of about 0.0001 per year for an eight-character password.

Referring back to why I started thinking about this in the beginning, we see that, in our case, we had seven hosts on the public net and experienced nowhere near that crack attempt rate, so this explains why we were not cracked. Our P_k was well below 0.006 [0.0001 x (7 hosts) x (8 years)], which is pretty good odds of not being broken into.

To digress for a moment, we and crackers have different points of view. From our point of view, we don't want to be cracked, so these are fairly good odds. However, from most crackers' point of view, they are also good

odds because he or she probably just wants to crack some system—not necessarily a particular one. So if he or she attacks 100,000 systems in an automated way, then one should burst open often enough to be interesting. This is kind of a sick win-win situation for both sides.

I don't want to have one of those systems, so I did the obvious thing—devise a way to detect crack attempts and cut them off. If I cut off hosts that attempt to crack my system more than, say, five tries in an hour, then the cracker needs to control a vast number of systems to be effective against me.

To do this, I wrote a short hack which monitors /var/log/secure to detect crack attempts on *ssh* and firewall-off suspicious hosts. The code requires Python 2.4, is GPL'ed, and could easily be adapted to monitoring for other types of crack attempts—say, against Apache servers. Again, see www.clove.com/clove_tech/download/.

Conclusion

Changing passwords frequently is a doomed strategy.

The important thing to do is keep P_k small by bounding the rate of growth of k and constructing passwords so that N is large, both of which are easy to do.

REFERENCES

[1] T. Perrine, "The End of crypt() Passwords . . . Please?" *;login:* (December 2003): 6–12.