Keeping Hackers at Bay

Improve password security to help keep your company's systems safe

By Robert Andrews

Since ancient times passwords have been used to prove identity, loyalty and membership. MIT's CTSS computer, introduced in 1961, is the first known use of passwords in computing. This system provided each professor with four hours computing time. In 1962, one professor was unhappy with his limited time, guessed a colleague's password and used the colleague's time as well. Hence, within one year of computer passwords, the first computer hacker was born. Since then, IT professionals have been in a constant race to make systems ever more secure from the growing number of would-be hackers.

Storing Passwords

In modern computing systems, including IBM i, passwords are never stored in their plain text or original version. Any listing or file that contained the actual user's password would be a ripe target and a major source of insecurity. Therefore, systems today store what is known as a one-way cryptographic hash. This one-way function takes in the password and returns a scrambled result, known as ciphertext. This function must be deterministic, meaning that given the same input, the same output will always be returned. This way, the system stores just the output of this function. Because the function is specifically designed to be one way, provided with the ciphertext, there is no way to reverse it and return the original plain-text password.

If two users selected the same password, they would get the same hash out of the function. To help with this, systems add "salt" or random values specific to each user. This way, even if two users happen to choose the same password, once combined with the salt and hashed, the results would be different. Rainbow tables, massive lists of pre-calculated hashes, were effective in cracking passwords before salt was added. When a user attempts to login, the password provided is combined with the user's unique salt and run through the same one-way function. The ciphertext result is compared to the stored ciphertext. If the two strings match, then the same input was used and the user is authenticated to the system.

Cracking Passwords

Because systems need to store the hash result, the first step is to extract the ciphertext version of the password. These may be centrally located in a single file or stored as part of each individual profile. Once extracted, the hacker has a target to shoot for. Two main ways exist for cracking passwords: dictionary attacks and brute force.

A dictionary attack is one where a present list of possible passwords is stored. Each of these possible passwords are salted, hashed and checked for a matching ciphertext. If they match, the result is found.

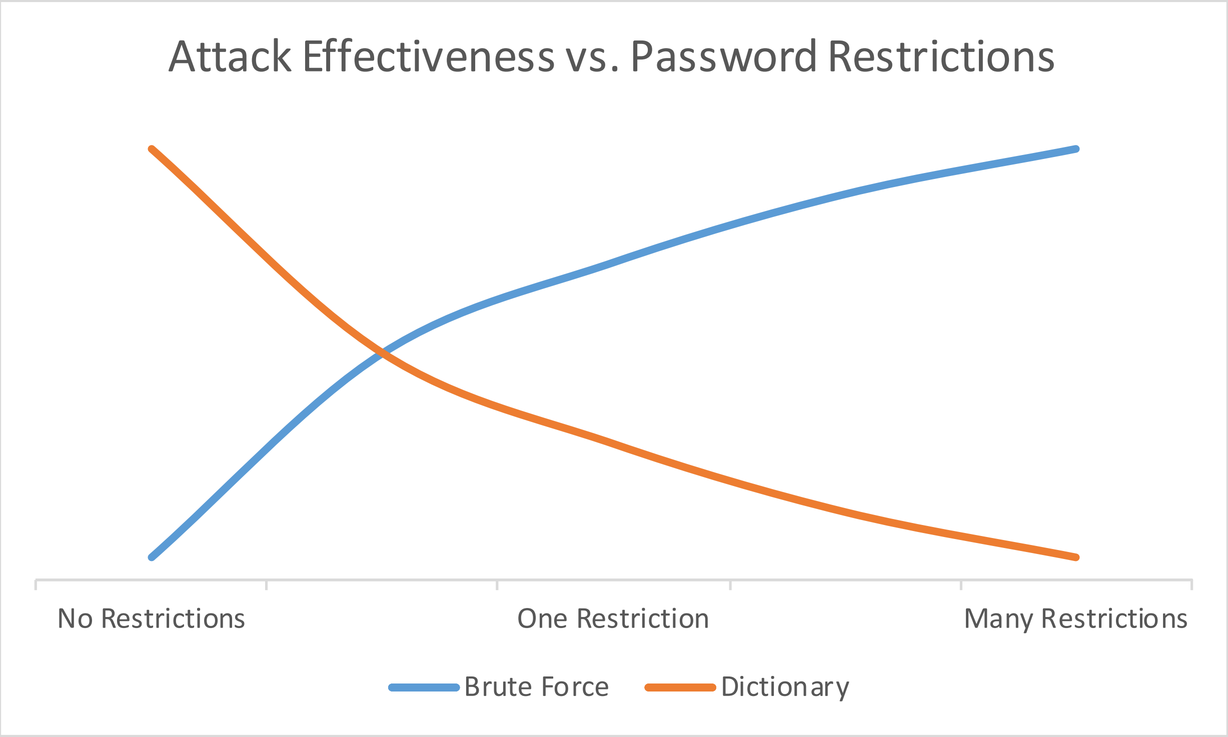
If not, the next possible password from the dictionary is attempted. As one might deduce, if the user's password is not in the list of possible passwords, it won't be matched and the account will still be secure. These dictionaries can be ordered by most popular passwords first in order to more quickly find a match. In addition, dictionaries can be customized based on the target. If the target's business, brand, sector, industry or location are known, the dictionary can be customized to include key words and phrases from those areas. For example, if the target password was from a company located in Wisconsin, sports teams and local city names could be added to the dictionary.

A brute force attack attempts to try all possible combinations of the input character set until a match is found. There is no need to start with a preset dictionary. What needs to be known is what characters are possible in the password and its length (specifically maximum length). For example, a cellphone has a four-digit passcode lock. The character set is only 10 possible values, 0 to 9, and four positions so there are 10^4, or 10,000 possible values. A brute force would start at 0000 and go to 9999 testing all 10,000 possible values until the proper code was found. If, instead, the phone had a four-character word, there are now 26 lower case letters and 26 uppercase letters for a total of 52^4 or 7,311,616 possible values, 731 times more! Increasing the number of different characters that can be used dramatically increases the number of attempts required to cover all possible values.

Preventing Attacks

Knowing how passwords can be attacked, defenses can be designed to counter these threats. To prevent against a dictionary attack, the password needs to be something that would not be in a common dictionary or a hacker's dictionary. To help with this, system administrators have placed in rules requiring items like digits (0 to 9) or special characters (!, @, #, etc.) be included in passwords. Sadly, too many people took an easy way around these rules and have informally developed what is known as leet speak (stylized 1337) where 3s would replace e's, 4s replace a's, !s replace i's, and so on. These substitutions have become so common that many dictionary cracking tools automatically try each word in its original form as well as doing a leet speak transformation on the word. This essentially doubles the effectiveness of the dictionary without needing to pre-generate all of those possible attempts. In addition to rules requiring non-character values, systems can also keep lists of the top common passwords and ensure they are not used in passwords.

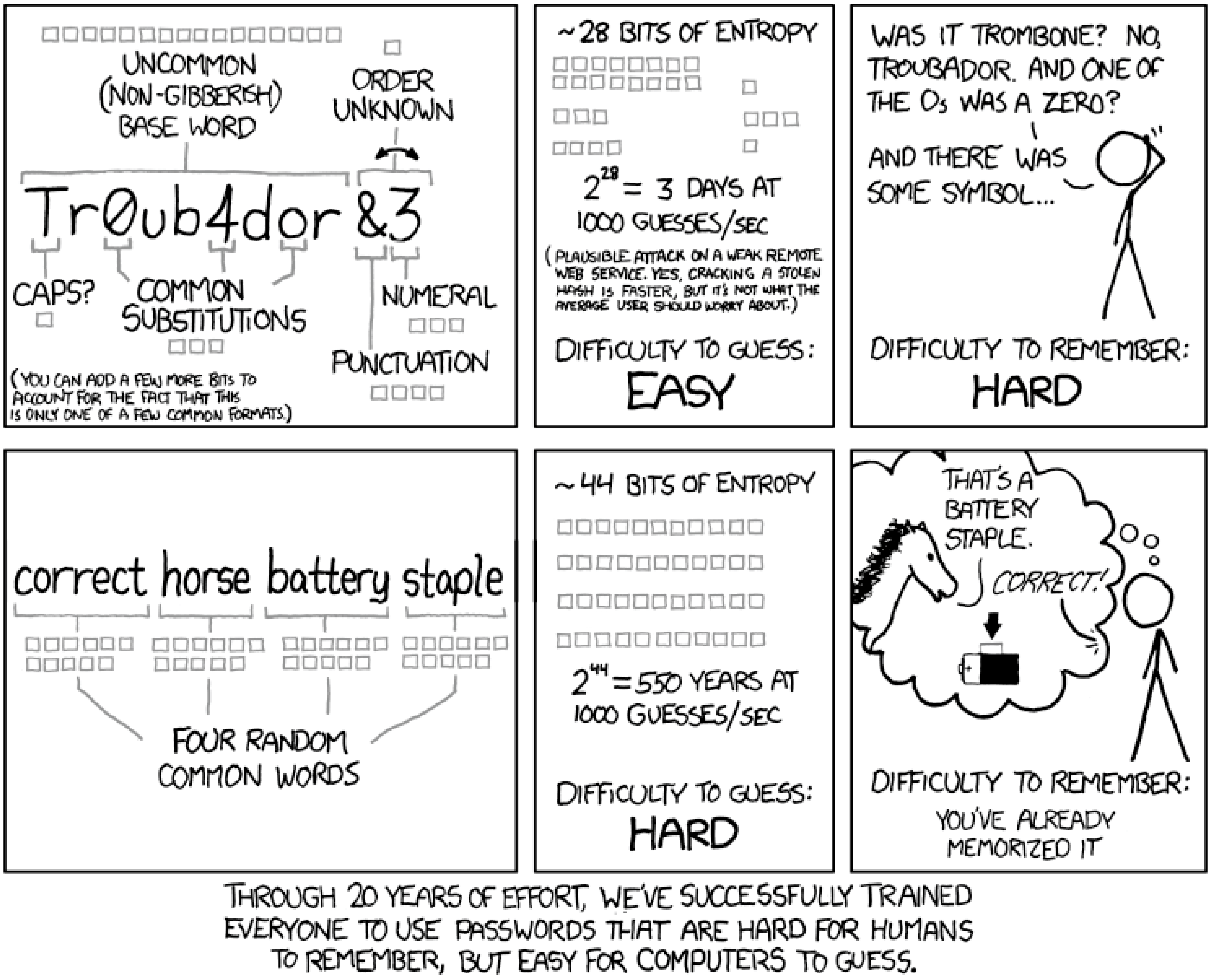
While these rules make a dictionary attack less effective, with brute force attacks they have the opposite effect. They do this by reducing the overall number of possible combinations that need to be attempted to cover the complete space of valid passwords. For example, assume an IBM i system running at password level 2 or 3. In this case, the system will use the password character set that is 26 lower case letters, 26 upper case letters, 10 digits, 32 special characters and the space for a total of 95 possible characters. If the password is eight characters long, a total number of 95^8 or 6,564,370,583,281,250 (6.56 quadrillion) total possible combinations exists. Now assume we add in a standard rule set that requires at least one digit, one character and one special character in the same password. Result is $52*10*33*95^5$or 132,780,808,875,000 (132 trillion) combinations. While this is still a lot, it's only 2.02 percent of the total possibilities. Therefore, the amount of time for a brute force attack is reduced 50 times over by putting in these standard rules (see Figure 1, page TK).



While this may seem like a large number, current computing hardware with multiple cores and advanced GPUs is shortening the amount of time required for brute force attempts from years to days. The advent of BitCoin has inadvertently given rise to other software that exploits the same underlying formulas for password cracking. Companies are also selling systems specifically designed for hacking with custom Linux software that handles the password cracking.

What is the Answer?

So with past attempts to stop hacking counteracting each other, what is the answer? Length. Long passwords are the best defense against today's attacks and they don't need to be complex. These long passwords are often called passphrases. They do not need to have strange characters or numbers substituted for letters. The main argument against long pass phases has been that they are hard to remember. This does not need to be the case! Many methods and techniques make long, yet easy to remember, passphrases. One way to develop a long password is to use a line from a song, movie or book. Spell it out, with spaces, capitalization, and punctuation, and it will be difficult for both dictionary and brute force attacks to find a match. Another method is to pick four words or items and just put those together. A classic geek cartoon from *xkcd* recommends "correct horse staple battery" as one such example. This is 16 times stronger than "Tr0ub4dor&3," which many may consider a strong password by today's rule environment. Another technique known as password haystacks involves repeating patterns to lengthen a password. In the password "Gr33nTr33s," by adding periods and colons, a small tree diagram can be made that more than doubles the length of the password to ".:..:.Gr33nTr33s.:..:." And finally, take advantage of password management tools that can generate and store very long, random passphrases. XKCD has a great comic around this topic (https://xkcd.com/936/).


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Recommendations

Based on the information presented in this article, review the password rules in place at your employer. Where did these rules come from? What was the intended result of the rules and do those still hold true with today's computing power?

IBM recommends simply increasing the minimum password length to at least 16 characters, regardless of any other rules in place. Consider adding tools to check when a user changes his/her password to make sure it's not on the list of most common passwords. The IBM Lab Services team offers one such tool that can be plugged into your IBM i system. Or have a password audit conducted. This service will reveal how many users are vulnerable through dictionary passwords and how many of those accounts have privileged or elevated access to the system.

For more information on these services or tools, visit the IBM i Security page (ibm.biz/IBMiSecurity).

Bio: Robert Andrews is a managing consultant specializing in security for the IBM Systems Hardware Client Technical Teams Lab Services Power Systems* Delivery Practice in Rochester, Minnesota. Besides security, Robert is an expert in DB2*, journalizing and DDM/DRDA. In addition to his technical work at IBM, Robert has been involved in emergency management and communications for almost a decade at all levels from local to federal. He has published seven books and holds degrees in mathematics, computer science, education and management. Email him at robert @robertandrews.com.

Sidebar: Password Recovery Caution

If a website offers password recovery where the site sends the user a password via email, that site is insecure! The site is either storing the password in plain text or using a reversible hash, neither of which is good. Sites should only offer password resets where the user can set a new password after some other form of verification, usually an emailed link or set of security questions.