Combating the Lazy User: An Examination of Various Password Policies and Guidelines

Sam Wilson

Combating the Lazy User: An Examination of Various Password Policies and Guidelines

Sam Wilson
Sept. 16, 2002

Abstract

A variety of password policies and guidelines are publicly available on the Internet. Most of them establish a set of rules which are either required or recommended for the user to follow when creating a password. Such rules include, but are not limited to, specifications for the length of the password, the character set(s) to be used, and whether or not dictionary words are allowed in the password. (A complete password policy also discusses many additional topics, such as how often passwords must be changed, but those additional aspects of password policies are not the subject of this paper.) This paper demonstrates that many published policies and guidelines will allow for the creation of weak passwords by lazy or inexperienced users. Such passwords may provide a relatively easy method of attack using custom dictionaries and readily available password cracking tools. This paper also makes recommendations by which the Security Administrator can improve the strength of the passwords which are created by the users on his system.

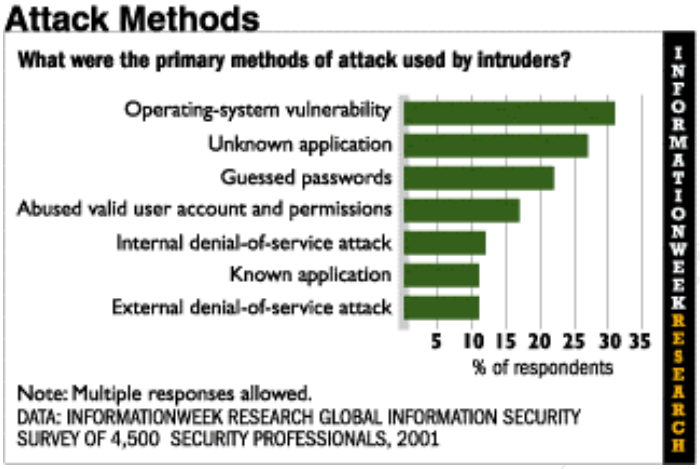
Background

It is common knowledge that weakly chosen passwords can provide an easy route through a system's defenses for an attacker. As noted by GeodSoft Website Consulting, "Passwords are important because they are still the primary key to most computer systems. At most sites, there is no greater opportunity for improving security with as small an effort than by adopting good password procedures." [1]

To illustrate the point, the graph shown in Figure 1, which was published in an online article by Information Week magazine [2], summarizes a survey which was performed in 2001. The responses, from 4500 security professionals, indicate that "Guessed Passwords" is the third most common attack method, occurring in 22 percent of the attacks.

For the Security Administrator to set a policy which demands strong passwords seems an obvious enough solution. One difficulty immediately arises in that the strongest passwords are random character strings which are notoriously difficult to commit to memory. As a result, the user forced to use a strong password is quite likely to write it down and stash it in some easily accessible location, such as the underside of the keyboard. Such action defeats the purpose for which the strong passwords were required. There are various schemes by which passwords which are more easily memorizable can be created. However, many

Figure 1: Results of Survey on Attack Methods [2]



such schemes tend to weaken the resulting passwords, so care must be taken as to how far one goes in that direction.

It can also be quite difficult to think up a good, strong password on the spur of the moment when the user is asked to enter a new password. If there are too many rules which should be applied, the user may not remember all of them, and is not likely to take the time to look them up. A new employee, in particular, may be required to create several passwords as he is first getting his system set up, but may not yet have read, or thoroughly understood and absorbed, his site's password policy. It follows that if the system itself does not enforce the rules stated by the written policy, by rejecting unsuitable passwords, it is very likely that many of the passwords selected by the users will not actually be in compliance with the policy.

Users will also strive to minimize the number of keystrokes required to enter their passwords. A famous quote from Perl language creator Larry Wall applies here, "the three great virtues of a programmer: laziness, impatience, and hubris." [3] Many password policies require at least one character each from three of the following four character sets: lower case letters, upper case letters, digits, and punctuation marks. The lazy or impatient user will comply with this policy by selecting exactly one upper case letter, and exactly one digit or punctuation mark, with the rest of the password consisting of lower case letters, to avoid using the shift key more often. Also, the characters which are not lower case letters are more than likely to be at the beginning and the end of the password. In most Western languages, sentences start with a capital letter and end with a punctuation mark. To construct a password in the same way is a very natural thing to do. An example of such a password might be "Tornado@". The number of shifted keystrokes is minimized, and while the policy has been complied with, the resulting password is nonetheless relatively weak.

Finally, some sites may require the user to create a password for each of a multitude of tools and programs. For example, at the author's work site, users must not only create a logon password, but must also use a second password to access e-mail, a third to use a calendar tool, a fourth for a discrepancy reporting system, a fifth for a configuration management program, and so on. Users are encouraged, but not required, to have unique passwords for each of the various tools. Most users are not so conscientious and use the same password for each tool. An attacker who gains knowledge of a logon password may then be able to access many of the tools, and has the potential to do more damage.

Strong Passwords

The strongest possible password policy would require completely randomized passwords, eight or more characters in length, drawn from the maximum available character set. The character set should not only include the 26 lower case letters (a-z), the 26 upper case letters (A-Z), and the 10 digits (0-9), but also should draw from the 33 special characters (the punctuation characters plus the space) available on a standard keyboard ('-!"#$%&()*,./:;?@[\]^_`{|}~+<=>).

But, as has been previously discussed, most strong, randomly generated passwords are such garbled strings that they can't be quickly or easily memorized, and so are not effectively useful. Table 1 shows a sample of random eight-character passwords, generated by the Automated Password Generator Online. [4]

Table 1: A Sample of Randomly Generated Passwords

co1Ks]v3	*m/<~~qc	KktOdUub	Zc[~>{vW	4IrdjCx|
V\y{%M_F	I:?_s5>	"6yxip"[	"Kl4"A"_	v`](%}S7

Of these, only "KKtOdUub" comes at all close to being readily memorizable, and it suffers from the fact that it actually includes no digits or special characters, and has two pairs of doubled letters.

Some systems will also allow selections from the set of non-displayable extended ASCII characters. For example, "Windows 2000 allows … passwords with several character types (letters, numbers, punctuation marks, and non-printing ASCII characters generated by using the ALT key and three-digit key codes on the numeric keypad)." [5] A table of the extended ASCII codes and the symbols which may be used to represent them can be found at URL: http://www.asciitable.com/. The use of these codes adds another 127 characters to the overall set which may appear in a password.

For an eight-character password which uses only lower case letters (or only upper case) there are 26^8, or about $2.09 * 10^{11}$ possible passwords. If a mixed case alphabet is used instead, there are 52^8, or $5.35 * 10^{13}$ possibilities. Table 2 shows the number of possible password strings for each of the cases and the

time required to test every possible password. The calculations assume a password length of eight characters and a processor capable of making 5,000,000 tests per second, such as is mentioned in a recent ZDNet article: "…no match for today's computers, which are capable of trying millions of word variations per second and often can guess a good number of passwords in less than a minute." [6] A similar statement is made in the UCLA Mathematics Department's password policy statement: "Hacker tools these days are so good that a brute force guessing program can break any 8 character lower case password. The shareware cracker that we use to check users' passwords can do it in thirty seconds, taking about 24 hours to do every user on PICnet." [7] The times here are computed for an exhaustive brute-force search.

Table 2: Cracking Times for Various Character Sets

Character Set	Size of Character Set	Possible Passwords (of length eight)	Time Required to Try all Possibilities (at 5 million/second)
Lower case letters	26	$2.09 * 10^{11}$	11.6 hours
Mixed case letters	52	$5.34 * 10^{13}$	123.7 days
Mixed case letters, digits, and special characters	95	$6.63 * 10^{15}$	42.1 years
All above plus extended codes	222	$5.90 * 10^{18}$	37,415 years

A general rule of thumb is that an acceptable password should not be capable of being broken by a brute-force attack in less than the expected lifetime of the password. An eight-character randomized password with mixed case letters will provide adequate protection only if the password is changed at least once every three months. A random password drawn from the 95 normal keyboard characters is sufficiently strong to resist a brute-force attack for most situations. Such a password which also includes some extended code characters certainly is strong enough to make a brute-force search highly impractical.

Published Password Policies

Let us now examine some published password policies and guidelines, to see how they compare to the standards set by truly strong passwords in the table above. Many published policies take into account the ease with which a password built entirely of lower case letters can be cracked, and specify that at least one or two characters are not lower case letters. However, seldom is anything beyond this minimum required, although additional recommendations may be made. The system will thus become susceptible to the lazy or inexperienced user, who complies with the letter of the policy, but chooses the weakest possible options. Such choices may open a door for an attacker.

It is assumed here that the attacker can discover the details of the target site's password policy, either through dumpster diving, social engineering, or because

he can gain physical access to the site, and can therefore craft an attack based on the expected tendencies of users under that policy. It is also assumed that the attacker is not targeting a specific user, but will succeed if he can obtain the password of any user on the system. If the site has a large number of users, a fair number of those users can naturally be expected to adopt the weakest type of password allowed under the password policy. If the attacker can guess the patterns that those passwords are likely to fall into, then he can use a Perl script or other means to build a "dictionary" tuned to those patterns. For example, LC4 (the latest version of L0phtcrack), a password auditing and recovery tool, "now supports multiple dictionaries for a single session; this allows the flexibility of specifying custom dictionaries for cracking sessions." [8] By using the specially tailored dictionary, the attacker has a much better chance of harvesting passwords from the system. He certainly won't get every password, but he can expect to do well enough to gain a foothold into the system, from which he can try other methods of attack. The security of the system depends on the weakest of the passwords.

In the following discussion, the following conventions apply for describing the patterns that passwords might fit into:

Table 3: Conventions for Password Patterns

A	Any upper case letter	a	Any lower case letter
9	Any digit	$	Any special character
#	A digit or a special character	x	A lower case letter or a digit

Example Policy 1

We begin with a rather simple set of password formation rules. David Milford's paper, "A System Security Policy for You", from the SANS Institute Information Security Reading Room, sets out a sample policy which includes the following two provisions:

- Alpha, numeric [sic] with at least one capital
- Minimum Password Length - 8 Alphanumeric characters [9]

Although digits are allowed under this policy, they are not required. The lazy user will select the weakest password that still complies with this policy, which will fall into the pattern "Aaaaaaa", (i.e., it will have an upper case initial letter, followed by seven lower case letters). A significant number of passwords can be expected to match this pattern. The attacker who tunes his cracking tool to search for one upper case letter (26 possibilities) followed by seven lower case letters (26^7 possibilities) could expect to successfully harvest a number of passwords from the system. The figures from Table 2 show that an exhaustive search of all passwords which fit the pattern should take less than half a day.

Example Policy 2

The phrasing in the second bullet from this policy published by Microsoft is very typical:

> The Passfilt.dll file implements the following password policy:
> - Passwords must be at least six (6) characters long.
> - Passwords must contain characters from at least three (3) of the following four (4) classes: English upper case letters, English lower case letters, Westernized Arabic numerals, Non-alphanumeric ("special characters") such as punctuation symbols [10]

This policy goes beyond the previous one in that at least three different character types are required, instead of two. But the minimum length of six characters specified causes severe damage. The password chosen by a lazy user of this system can be expected to fall into the pattern "Aaaaa#", (i.e., it will start with a capital letter, contain four lower case letters, and end with a digit or special character). The total number of passwords which fit this pattern is $26 * 26^4 * 43$, or about $5.11 * 10^8$. Under the assumption that a tailored cracking program can run five million tests per second, these passwords will be exposed in under two minutes.

Example Policy 3

Another sample policy is available from the resource area of the SANS Security Policy Project:

> Strong passwords have the following characteristics:
> - Contain both upper and lower case characters
> - Have digits and punctuation characters as well as letters
> - Are at least eight alphanumeric characters long. [11]

This policy is strengthened compared to the previous one by the specification of a length of at least eight characters. It does not, however, specify any number of upper case letters, or of digits and special character, so most users will stick to the minimum of one of each. As with the previous policy, there are likely to be many passwords created of the pattern "Aaaaaaa#". The comparative metric for these passwords is $26 * 26^6 * 43$, or about $3.45 * 10^{11}$. The hypothetical cracking program will require about just over 19 hours to reveal these passwords.

Example Policy 4

The following guidelines from M-Tech Mercury Information Technology, Inc**.** are strengthened by the requirement for at least two digits, which not to be at the beginning or end of the password. But they are also weakened by the

specification of a minimum length of six characters and by not requiring any special characters.

> Your password must:
> - Have at least 6 characters
> - Have upper and lower case characters
> - Have at least three letters
> - Have at least two digits
> - Have at least two digits not at the beginning and end [12]

The most likely pattern to be used in this case is "Axxxxa". The number of possible passwords which fit this pattern is $26 * 36^4 * 26 = 1.14 * 10^9$. The time required to try all combinations for this pattern is under four minutes.

Example Policy 5

The policy implemented on Unix systems at the University of Waterloo contains the following rules:

- Maximum Length: Eight (8) characters.
- Minimum Length. Seven (7) characters.
- Unique Characters. At least five (5) different characters.
- Character Types. Characters from at least three (3) different character types -- upper case, lower case, digits, punctuation, etc.
- Long Alpha Sequences. No alphabetic sequence any longer than three (3) characters.
- Long Digit Sequences. No digit sequence any longer than two (2) characters. [13]

The pattern that the lazy user might tend to choose for passwords under this rule set is a little more difficult to determine, due to the requirement that passwords have no more than three consecutive alphabet letters or two consecutive digits. Some password patterns which fit these rules and are likely to be commonly used are "Aaa9aa$", "Aa9aaa$", "A99aaa$", "Aa99aa$", or "Aaa99a$". The number of possibilities for the first two of these is $26 * 26^4 * 10 * 33 = 3.92 * 10^9$, while for the final three it is $26 * 26^3 * 10^2 * 33 = 1.51 * 10^9$. The overall number of passwords to test is thus $(2 * 3.92 * 10^9) + (3 * 1.51 * 10^9) = 1.24 * 10^{10}$. A password cracking program tailored to these patterns could search through all possible combinations in under 42 minutes.

Example Policy 6

The sample password policy published by the Texas State Library and Archives Commission includes a recommendation which is rather common:

> shorT#duck — this password also has a mix of three of the categories mentioned. Notice there are two unrelated words joined together, but with mixed case and with a special character between them. Joining two words this way also helps you remember your password. [14]

A similar example is published by the University of Pennsylvania: "Suggestions: take two unrelated short words joined by a special character, such as Big$Deal, …" [15] In this case the capitalization is done at the beginning of each included word, which is essentially equivalent to not using capitalization at all. (It is also somewhat difficult to accept the claim that "big" and "deal" are unrelated words.)

The statement in the Texas State Library policy is correct in that such passwords are easier to remember. However, the following analysis shows that the advice is not adequate. Since real words are being used, these passwords are susceptible to a modified dictionary attack.

First we will compute the size of the set of possible passwords. The Unix words list, /usr/dict/words, (on the author's machine) contains approximately 800 three-letter words, 2200 four-letter words and 3200 five-letter words. Almost all common English words are represented in this word list. Next, we note that each three-letter word can be represented in eight different ways by changing the case of the letters, e.g., "abc", "Abc", "aBc", "abC", "ABc", "AbC", "aBC", "ABC", Likewise, each four-letter word can be represented 16 different ways, and each five-letter word 32 different ways.

A password constructed according to these suggestions could be put together in several combinations, a three-letter word followed by a five-letter word, for example, or a pair of four-letter words. The attacker could write a Perl script which would read all three-, four-, and five-letter words from /usr/dict/words or a similar dictionary, and combine those words in all possible ways, with each possible special character and digit appearing between the two words. The attacker would use the output file as the custom dictionary for his cracking program. Table 4 shows the results of a few minutes work with a spread sheet. The numbers of possible passwords, along with the times which would be required to search exhaustively through them, were computed. The resulting search times are insufficient to provide adequate protection for the system.

An improvement to this scheme would be to place extra special characters into positions which break up the words. For example, using the University of Pennsylvania advice as a starting point, a user might create the passwords "Fish&Book" or "Inch5Games". These weak passwords could be replaced with "Fi<sh!B?ook" or "I~nch5Gam(es". The replacement passwords are only slightly harder to memorize than the previous versions, but are quite a bit harder to crack, because the modified dictionary attack described above will no longer succeed.

Table 4: Numbers of Combinations for "Word$Word" Forms

Combination	Number of combinations, all lower case or only first letter capitalized	Number of combinations, fully mixed case
3+3	$800 * 800 * 43 = 2.75 * 10^7$	$2.75 * 10^7 * 8 * 8 = 1.76 * 10^9$
3+4 or 4+3	$2 * 800 * 2200 * 43 = 1.51 * 10^8$	$1.51 * 10^8 * 8 * 16 = 1.94 * 10^{10}$
3+5 or 5+3	$2 * 800 * 3200 * 43 = 2.20 * 10^8$	$2.20 * 10^8 * 8 * 32 = 5.64 * 10^{10}$
4+4	$2200 * 2200 * 43 = 2.08 * 10^8$	$2.08 * 10^8 * 16 * 16 = 5.33 * 10^{10}$
4+5 or 5+4	$2 * 2200 * 3200 * 43 = 6.05 * 10^8$	$6.05 * 10^8 * 16 * 32 = 3.10 * 10^{11}$
5+5	$3200 * 3200 * 43 = 4.40 * 10^8$	$4.40 * 10^8 * 32 * 32 = 4.51 * 10^{11}$
Totals	$1.65 * 10^9$	$8.92 * 10^{11}$
Time to try all combinations at $5 * 10^6$/sec	5.5 minutes	just over 2 days

An additional step which would result in even more improvement is to also arbitrarily change the case of the alphabet letters, e.g., "fi<Sh!B?Ook" or "I~NCh5gAm(Es". For comparison with the previous results, the number of possible passwords which can be formed in this way from two common four letter words, using mixed case letters, with a special character or digit between the two words, and two more special characters or digits embedded in an arbitrary position within each word, is $2200^2 * 43 * 43^2 * 9 * 16^2 = 8.87 * 10^{14}$, which would result in a search time of 5.6 years. Allowing for three- and five-letter words would extend that search time significantly.

Another variation on this password creation scheme is found in the password policy statement from the Division of Information Technology Management of the State of New Hampshire.

> Algorithms can be developed to help users remember mixed character passwords. The following is an example of an algorithm that could be used:
>
> 1. Think of two words easily remembered – airplane wing (for example)
> 2. Add up the number of characters – 12
> 3. Take out the vowels and put the number in between – rpln12wng
> 4. Capitalize the first and last word – Rpln12Wng
> 5. Change one of the letters to a special character – Rp!n12Wng
> 6. You now have a secure password that is somewhat easy to remember. [16]

This algorithm does produce a fairly strong password. The exclusion of vowels prevents the password from falling victim to a dictionary search, while the inclusion of a special character in an arbitrary position broadens the range of possible patterns which must be searched. The method could be made even

better by arbitrarily shifting letters from lower to upper case, instead of capitalizing the first letters of the two words.

On a system which limits password length to eight characters, the "word$word" method of password creation should be avoided entirely, not only because there is no room to add extra non-letter characters, but also because shorter words must be used. If the user added two-letter words to the scheme, his choices would be combinations of 2+5 or 3+4. Combinations of 3+5, 4+4, or 4+5 would result in passwords too long to use, while combinations of 2+2, 2+3, etc., would yield passwords shorter than eight characters.

Pronounceable Passwords

Another bit of password advice which can be commonly found is this: "alternate between one consonant and one or two vowels, up to seven or eight characters; this creates nonsense words that are usually pronounceable and, therefore, easy to remember." [17] There are a number of password generation tools available which will produce random, pronounceable passwords. For example, the sample passwords displayed in Table 5 were generated online by the Java Password Generator. [18]

Table 5: Sample Automatically Generated Pronounceable Passwords

tealdomi	illespic	atmorooh	ramsfore	witaingl
eradbrap	adenspan	psabifeu	eseinsev	uallaher

As it stands, without further modification, this advice is very poor, because the resulting passwords all fit the pattern "aaaaaaaa". As has already been shown, passwords of this pattern can be cracked in a matter of hours. Only if the users are required to make some of the letters upper case, and to add digits and special characters, would these passwords be acceptable. Another alternative would be to insist that passwords of this form contain at least ten characters. A password of the form "aaaaaaaaaa" could take almost a full year to crack using a brute-force search, rather than twelve hours.

Passwords Derived from Phrases

A technique which is frequently described in compilations of password advice is to start from a memorable phrase. The CERN Security Handbook suggests, "Choose a line or two from a song or poem, and use the first letter of each word. For example, 'In Xanadu did Kubla Kahn a stately pleasure dome decree' becomes 'IXdKKaspdd'. [19] In this sample there is unfortunately no addition of digits or special characters, and the capitalization is even left unchanged from the original phrase, all of which weakens the password. At least the password is long enough, at ten characters, to still retain a degree of strength.

A more up-to-date version of the technique is well described in this excerpt from the University of Colorado's published security guidelines:

> The best strategy is to transform a phrase into a password. Think of a phrase, then use the first letter of each word and use numbers or symbols to represent some words. For example, the phrase "Thank goodness that I have work!" could become Tg4Ihw! The phrase "When it quacks, pound it!" becomes Wiq#i! Lines from children's songs (Yankee Doodle went to town = !Ydw2t), statements about food (I hate canned green beans! = Ih!cgb), statements about places (My heart belongs to San Francisco = Mhb2!SF) can be easier to remember. Notice that the punctuation marks are put in non-standard places in the phrase. [20]

If the user is careful and creative enough, this advice will result in a strong password. A good phrase should have personal meaning to the user and should not be a well-known phrase from literature or popular culture. However, it takes time and effort to think up such a phrase. Many users may not want to spend an adequate amount of time on it. The lazy user is more likely to pick a phrase which relates to current popular culture, such as a recently released movie or song title, a recent news event, or a jingle from a television commercial. An attacker who knew that the users at his target site were encouraged to create passwords in this way could spend some time generating variations of likely phrases and might very well have some luck with this approach.

It should also be pointed out that the sample passwords given in the excerpt from the University of Colorado guidelines, in all but one case, start with a capital letter. Further, with the exception of "SF" for "San Francisco", the initial capital letter is the only upper case letter in the password. If the author of this advice fell into that trap, so too will many users who follow this advice. Finally, in each sample password, there are at most two non-letter characters. These facts suggest that the following patterns could be used to generate a tailored dictionary which would result in some success against these passwords: "Aaaaa#a#", "Aaaa#aa#", … , "A#a#aaaa". There are fifteen such patterns, (assuming an eight character password) and the number of possible passwords given by each of them is $26 * 26^5 * 43^2 = 5.71 * 10^{11}$. The resulting tailored dictionary would then have $8.57 * 10^{12}$ entries. It would take about twenty days to check all possible entries which matched these patterns, assuming, as usual, a program capable of checking five million words per second. As in earlier cases which were discussed, a balance between upper and lower case letters would make this scheme much stronger.

Passwords for Multiple Systems

As was mentioned earlier, many sites require a user to create passwords for more than one system, or require passwords for individual programs within the

system. This fact can present a security problem since the typical user is likely to want to memorize as few passwords as possible, so will use the same password everywhere. The situation may be made worse due to the fact that the user is almost certain to also have password protected accounts on his home system and on the Internet. A determined attacker might snoop for lightly protected passwords on an employee's home system, then try those passwords to break into the employee's machine at the work site. The employee should certainly be instructed not to duplicate passwords between his home system or on the Internet, and at work.

The computer staff at the Stanford Linear Accelerator Center give this advice in answer to the question "How many passwords should I have?":

> If someone does obtain your password, they have access to all of your accounts on all systems. This is one of the primary ways that hackers jump from system to system and site to site. … Accounts at SLAC should have different passwords from accounts at other places. You especially should not have the same password on an online service provider as you do on your SLAC account. Passwords for web-based services and other Internet services are generally not very secure and shouldn't be reused for your SLAC accounts. [21]

One way to surmount the problem of too many passwords would be to start from a good, strong, base password, and to modify it in some way to make it at least slightly different for each place it is to be used. For example, if the user derives a base password from "dime" and "birds" to get: "di*Me@BiR%ds", he might then alter it by adding "_E" to the end for the e-mail program, adding "_C" for the calendar program, and so on.

Password Generators

Password generators have been mentioned twice before in this article. There are a fairly large number of them to be found on the Internet, in a variety of flavors: Windows, Unix, or Mac; freeware, shareware, or commercial; online, downloadable, or shrink-wrapped. A user who is willing to spend the time to do the research to find an acceptable product could benefit from its use. There are a few points to be made, however. As the sample output in Table 1 shows, the randomly generated passwords are very difficult to memorize. One way to cope with this difficulty is to have the generator produce a large list of passwords, and then to look through the list to find the one which looks the best to use. The danger here is that the password which is easiest to memorize may also be the weakest password on the list. The user should be careful to avoid passwords which contain substrings which are words, or which do not contain the full range of character types.

If a password generator does not generate completely random passwords, they are very likely not to be as strong as they need to be, as the sample passwords in Table 5 illustrate. In this case, the user should take the generated password as a starting point and modify it by changing the case of letters arbitrarily, and by adding digits and special characters, to arrive at an acceptably strong password.

Finally, the online password generators should be used very carefully, if at all. The possibility for compromise is present in several different ways. If the owner of the site is unscrupulous, he may secretly store the generated passwords on his server and track the computer from which the request came. Later he may try to attack that computer using the passwords which were passed to it. In addition, the generated passwords are very likely to be transmitted in the clear, leaving them vulnerable to network sniffing by a third party. Finally, the generated passwords are in essence already written down, since they are temporarily stored, unencrypted, in the memory or hard drive of the requesting computer. If the user does not clear the browser's cache and any other location to which the password was stored, the passwords may be open to theft.

The dangers of an online password generator are perhaps most vividly illustrated by a website titled, "Password Generation Service." The visitor is asked to enter his name, his machine name, and the machine's IP address. No matter what information is entered, when he hits the "Select" button he is shown the following message:

> Based on the information that you supplied, we recommend using the password "Iaacitut". This password has the advantage that it can be remembered with the simple mnemonic
>
> **I a**m **a c**omplete **i**diot **t**o **u**se **t**his [22]

Recommendations

The Security Administrator should strengthen his password policy so as to explicitly prohibit a user from using a password which matches the most easily guessed patterns.

- The password policy should prohibit passwords which are words or contain substrings which are words.
- The policy should not merely suggest, but should require a balanced mixture (more than just one of each) of upper case and lower case letters, digits, and special symbols.
- The policy should require frequent shifts from one character type to another, avoiding long substrings of lower case letters in particular.

- The user should be strongly discouraged from using a password which starts with a capital letter and ends in a digit or special character, especially if the remaining characters are all lower case letters.

- The user should be encouraged to properly apply one or more of the published techniques to produce a password which is both strong and fairly easy to memorize.

The user should be required to avoid using a password which he also uses on his home computer or for an Internet service.

If a user must maintain several passwords for different tools on a single system, he should be required to use a different password for each tool.

The password policy should be written and published so every user is aware of the requirements. Each user should sign a form to indicate that he understands and agrees to comply with the policy. A new user, in particular, should be made to read and comprehend the policy before beginning to work on the system.

Whenever possible, the password policy should be enforced by software which checks passwords as they are entered and rejects passwords which do not meet the standard set by the policy. When this is not possible, a regular audit should be performed by the Security Administrator, by running password cracking software against the system, to check that the passwords being used do meet the requirements.

On a system where a user may incorporate extended ASCII codes into his password, he should be made aware of that fact. He should also be taught how to enter the codes, and should be encouraged to use them.

Each user should be educated as to the advantages and limitations of using password generation software, especially the dangers of online password generators. The Security Administrator should investigate the available software and consider making an appropriate program available to the users on his system.

Summary

Many published password policies and guidelines will result in weak, poorly constructed passwords if the user is not conscientious about applying the guidelines properly. In such cases, an attacker can easily determine the expected patterns into which many passwords will fit. He can then tailor a custom dictionary to produce all strings which match those patterns and feed that dictionary to a password cracking program.

On the other hand, there are many good password suggestions which can help the user to create a strong password, and to avoid the mistakes which result in severely weakened passwords. The Security Administrator should find or create an appropriate list of such suggestions, tailor it to his site, and write the suggestions into his published password policy. He should then make sure that the policy is understood by all users and enforced on the systems for which he is responsible.

References

[1] GeodSoft Website Consulting. "Good and Bad Passwords How-To." URL: http://geodsoft.com/howto/password/. (15 Sep 2002).

[2] Hulme, George V. "It's time for developers to think and act differently." Software's Challenge. Information Week, 21 Jan 2002. URL: http://www.informationweek.com/story/IWK20020120S0003. (13 Sep 2002).

[3] Wall, Larry, Tom Christiansen, and Randall L. Schwartz. Programming Perl. Second edition. Sebastopol, CA: O'Reilly & Associates, Inc., 1996. p. xiii.

[4] Mirzazhanov, Adel I. "Automated Password Generator Online." 12 June 2002. URL: http://www.adel.nursat.kz/apg/online/index.php. (14 Sep 2002).

[5] Microsoft TechNet. "Windows 2000 Server Baseline Security Checklist." Microsoft, 2002. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/chklist/w2ksvrcl.asp. (15 Sep 2002).

[6] Lemos, Robert. "Psst… I know your password." ZDNet News, 22 May 2002. URL: http://zdnet.com.com/2100-1105-920092.html. (14 Sep 2002).

[7] Mathematics Computing Group. "Password Policy." UCLA Mathematics Department, 17 Jan 2001. URL: http://www.math.ucla.edu/computing/user_support/policies/password.html. (14 Sep 2002).

[8] @stake, Inc. "LC4: What's New." 2002. URL: http://www.atstake.com/research/lc/whatsnew.html. (15 Sep 2002).

[9] Milford, David. "A System Security Policy for You." Information Security Reading Room, SANS Institute, April 25, 2001. URL: http://rr.sans.org/policy/sys_sec.php. (13 Sep 2002).

[10] Microsoft Product Support Services. "How to Enable Strong Password Functionality in Windows NT." Microsoft Knowledge Base Article - Q161990. Microsoft, 11 June 2002. URL: http://support.microsoft.com/support/kb/articles/Q161/9/90.asp. (15 Sep 2002).

[11] The SANS Security Policy Project. "Password Policy." URL: http://www.sans.org/newlook/resources/policies/Password_Policy.pdf. (14 Sep 2002).

[12] M-Tech Mercury Information Technology, Inc. "Password Strength Rules." URL: http://www.psynch.com/security/policy-details.html. (14 Sep 2002).

[13] Information Systems and Technology, "UWaterloo Password Policy." University of Waterloo, 25 May 2000. URL: http://ego.uwaterloo.ca/~uwdir/policy/Passwd.html. (13 Sep 2002).

[14] Texas State Library and Archives Commission. "A Sample Password Policy." 15 Oct 2001. URL: http://www.tsl.state.tx.us/ld/pubs/compsecurity/ptthreepwdpol.html. (15 Sep 2002).

[15] School of Arts and Sciences. "How secure is my account and password?" University of Pennsylvania, 28 May 2002. URL: http://www.sas.upenn.edu/computing/help/Server/account-security.html. (15 Sep 2002).

[16] Towle, Thomas. "Computer Password Standard." State of New Hampshire Department of Administrative Services, Division of Information Technology Management. URL: http://admin.state.nh.us/ditm/computerpasswordst.doc. (15 Sep 2002).

[17] Kessler, Gary C. "Passwords - Strengths and Weaknesses." Internet and Internetworking Security. Ed. J.P. Cavanagh. Auerbach, 1997. URL: http://www.garykessler.net/library/password.html. (14 Sep 2002).

[18] Van Vleck, Tom. "Java Password Generator." 31 Jul 1997. URL: http://www.multicians.org/thvv/gpw.html. (15 Sep 2002).

[19] Cons, Lionel. "Passwords." CERN Security Handbook, Version 1.2. European Organization for Nuclear Research, 12 Dec 1996. URL: http://consult.cern.ch/writeups/security/security_3.html#SEC10. (15 Sep 2002).

[20] Winsor, Lindsay. "Good Security Practices for Our Users." University Management Systems, University of Colorado, Dec 2001. URL: http://www.cu.edu/~security/users/access/goodprac.htm. (14 Sep 2002).

[21] Johnson, William B., John Halperin, and Bob Cowles. "Suggestions for Selecting Good Passwords." 28 Feb 2002. URL: http://www.slac.stanford.edu/comp/security/password.html. (15 Sep 2002).

[22] DigiCrime. "Password Generation Service." URL: http://www.digicrime.com/passwd.html. (15 Sep 2002).