Kristen Kurimoto
CS 181 JT – Computer Security
Erlinger/Bull
13 April 2006

How effective are current password policies in protecting against cracking attempts?

Text-based passwords are a ubiquitous form of protection, providing a means of authentication and authorization for many systems. However, it is unclear whether a password is an adequate mechanism of protection. A very weak password can compromise the security of the whole system, but even strong passwords are vulnerable to attacks. In this paper we examine typical cracking techniques performed by popular password cracking programs to discover characteristics of weak passwords and analyze the effectiveness of current password policies and recommendations. Not only must users choose strong passwords, but system administrators must have a well-defined policy for managing passwords. We conclude that while textual passwords have many flaws, because they will continue to be widely used into the foreseeable future, enforcing a good password policy becomes very important for security.

The purpose of password crackers is to discover a legitimate password in combination with a username in order to gain access to an account (most notably, root on UNIX, or an administrator account on Windows) or a system. Malicious users may use password crackers to gain unauthorized access. If they perform the cracking process undetected, then they can potentially damage the cracked account or launch an attack against the system before anyone suspects a problem. On the other hand, password crackers can also provide proactive system administrators a means of enforcing password policies and detecting weak passwords before an attacker can compromise the situation. Nevertheless, the widespread availability and use of password crackers again comes

down to a question of policy: if system administrators choose to run password crackers, how should they respond to the results of the crack attempt? Most often, they will require users to change their passwords, but they may need to reexamine the policy itself. We will further explore this facet of policy after we describe the password crackers themselves.

Rather than trying to reverse the password hash or decipher the ciphertext, password crackers most often compare the hashed values stored in the password file to the hash of the guess. If the two hashes match, then the password is compromised. Many password crackers are available for use on UNIX and Windows. For illustrative purposes, we will focus on John the Ripper, a widely used cracking program that works with multiple operating systems. First released in 1996 [2], John provides a variety of utilities in a powerful and flexible interface. It runs on multiple operating systems, accepts multiple password file formats, understands several password hashes, and allows the user to customize the attack.

To compute possible passwords, password crackers employ two main techniques: dictionary attacks based off wordlists and brute-force attacks. Dictionary attacks can efficiently crack weak or moderately weak passwords. The cracker hashes each word in a wordlist and compares it to the password file. While seemingly simplistic, dictionary attacks are effective because many users choose weak passwords, including personally meaningful words, such as the names of family members or pets. In fact, in two independent surveys [4,15], half of the participants reported using personally meaningful words as their password. This means that their passwords are most likely easily cracked using a wordlist. Wordlists are openly available on the internet at very low prices; the

website for John the Ripper offers wordlists in over twenty languages for fewer than thirty dollars [10].

Brute-force attacks run more slowly, but enable the discovery of stronger passwords. By definition, a brute-force attack attempts to find every combination of characters in the input space; clearly, the larger the input space, the longer the brute-force attack takes. The limitation on brute-force attacks comes primarily from hardware speed, but this limitation is weakened as hardware speeds increase. Moreover, hardware restrictions can be partially overcome by distributing password generation across multiple computers. Therefore, it is all the more important for users to choose good passwords from a large input space of characters.

Hybrid attacks are powerfully deployed in numerous password crackers to combine the coverage of brute-force attacks with the efficiency of word lists. For example, John has enables users to create rules to modify and permute the characters in the words in a wordlist, thereby multiplying the number of guesses without resorting to a full brute-force attack. This hybrid attack may be especially effective if the attacker knows certain requirements that are integrated into the password policy. For instance, if a policy specifies a minimum and maximum length of acceptable passwords, John can be configured to append or prepend characters to achieve the desired lengths and reject words whose lengths are outside of the bounds. The rules are also extensive enough to account for many common strategies that people employ when creating passwords, such as replacing characters with numbers or symbols, reversing words or rotating the characters in a word, or using only uppercase or only lowercase letters. John's rule preprocessor further expedites rule creation by allowing attackers to write one command

that will be expanded into multiple rules [11]. If the attacker wanted to append two numeric digits to the end of each of the words in the wordlist, he or she could write the "$[0-9]$[0-9]" rule, where $ indicates the append command, instead of writing out 100 different rules. For very little effort, John provides high returns.

Attacks can be carried out offline if the password file is available, or online for real-time access. Clearly a system administrator has access to the password file, and malicious users can obtain it through various means. Since protecting the password file itself is more of a problem with data protection as opposed to the implementation of a good password system, we will not focus on the logistics of obtaining the file. Online attacks provide a different challenge. One possible protection against online attacks is to lock out a user after a password fails a given number of times. However, Pinkas and Sander point out that such a strategy results in high customer service costs and vulnerability to denial of service attacks [13]. Therefore, the most cost-effective way to ensure that passwords remain protected is to choose a good password to begin with.

We now examine various suggestions and policies for creating passwords and evaluate whether the suggestions address the vulnerabilities that password crackers expose. We gathered a sample of data from publicly available resources, mainly websites of universities and colleges. One drawback to this method is that it is unclear whether the systems in question enforce the posted guidelines, or whether the guidelines are merely suggestions. The most striking aspect of this investigation is the lack of consistency in the policies. While most policies did not contradict each other and are written in the same spirit, there were definite variations in the required length, required characters, forbidden sequences or passwords, and expiration date. These inconsistencies make it

even harder for the average user to remember how to create a strong password, let alone remember the passwords themselves.

The length of a password can make a great difference in the success of a brute-force attack. Unsurprisingly, most sources (11 of the 15 studied) recommended a specific length for passwords. Table 1 summarizes the suggested password lengths (if any) found in the sources. If there are 72 allowed characters in four different character classes for each position in the password (26 lowercase letters, 26 uppercase letters, 10 digits, and 10 special characters), then the total number of possible passwords in a six character password is $72^6 = 139,314,069,504$. According to the *Anti-Hacker Tool Kit*, John can compute about 400,000 cracks per second [16], so the total time needed to crack the entire password space is (72^6 words / 400,000 cracks per second) / 86,400 seconds per day = 4 days. This is a reasonable amount of time, even running on one computer. Consequently, allowing users to have a password as short as six characters presents an unnecessary vulnerability to the system. For a seven character password, the amount of time needed is (72^7 / 400,000) / 86,400 = 290 days, which seems adequately long enough to protect against an attack. However, we must then consider the possibility (and likelihood) of distributing password generation, as mentioned earlier. Essentially, each node receives a unique, incremental starting value from which to compute possible

Length Suggested	# of Sources	Sources
at least 6	4	MIT, UCSD INNOPAC, University of Colorado at Boulder, Scripps
at least 7	2	Stanford, *Surviving Security*
at least 8	3	Geodsoft, Tufts, Penn State
exactly 8	2	Harvey Mudd, UC Berkeley
no recommendation	4	UCLA, Pomona College, Security Focus, *Computer Insecurity*

Table 1: Suggested Password Lengths by Source

passwords, allowing the hacker to divide the work roughly evenly among the nodes. Thus, if a cluster of thirty computers were to work on cracking passwords, it would take fewer than ten days to crack the entire input space. An attacker would have to spend significantly more time and money in order to successfully and completely carry out an attack against a system with seven character passwords. Nevertheless, these resources are still in a reasonable range, and it is certainly possible for a determined attacker to compromise such a system. Undoubtedly, requiring seven characters provides more security than requiring six characters, but this length does not completely protect against a crack. Lastly, cracking passwords with eight characters on UNIX would entail (72^8 / 400,000) / 86,400 = 20,897 days = 57.25 years worth of work. The attacker would have to distribute across hundreds of machines in order to discover any significant percentage of the eight character password space. Therefore, a password with at least eight characters chosen from the four classes of characters appears to be the best defense against a brute-force attack. Note, however, that Windows NT's password hashing scheme ignores letter case and also splits passwords greater than seven characters in length into a seven character part and a residue part. If the original password is eight characters long, the password file will contain the hashed value of a single character, virtually guaranteeing that these characters will be discovered during the attack. As such, an eight character password is no more secure than a seven character password on Windows NT.

Since dictionary words are the heart of the wordlists used in password crackers, it seems that prohibiting dictionary words as passwords must be a priority. However, only eight of the fifteen sources explicitly state that dictionary words are poor passwords.

Furthermore, the question again arises as to whether these policies are enforced. If the policies are not enforced, then the systems are just as vulnerable as those that do not ban dictionary words outright. The increased danger in the latter case is that users may not even know that dictionary words are weak passwords and may choose one believing it to be secure. Password policies should inform users about how to choose a secure password, so omitting a ban on dictionary words seems like an egregious mistake. Additionally, only four sources (Geodsoft, MIT, Stanford, and Security Focus) specify that replacing letters with symbols, such as replacing “S” with “$” or “E” with “3,” does not increase security. Even if the wordlists do not already include these common substitutions, an attacker can trivially set up a hybrid attack in John or other crackers using built-in substitution rules. Alarmingly, Harvey Mudd’s policy actually encourages this kind of substitution, stating that "G00d0n3!" is a good password. While this password is at least composed of two dictionary words, rather than one, and has a special character appended to the end, it utilizes the common substitutions for the letters “o” and “e.” The further implication is that users may take a single, longer dictionary word and perform common substitutions. As an example, the system would accept “Re$34rch,” though it is not very secure. All password policies should be modified to include these known facts of the dangers of dictionary words and dictionary words with common substitutions.

Many password policies also place restrictions on passwords at the character level. One type of restriction requires the use of characters in a particular class. The running times analyzed above assumed that each position in the password drew from a large set of characters. Seven of the fifteen sources studied did in fact require or

recommend using at least one character from each of the four classes of characters (Andress, Furnell, Security Focus, Harvey Mudd, MIT, Stanford, and Tufts). Other schools have guidelines that are not as stringent. For example, Penn State and the UCSD INNOPAC system only require one letter of either case and one number. Conceivably, if the passwords were checked upon creation, a weak password such as "aaabbb11" would pass muster. This means that an attacker could initially restrict the search space to much smaller dimensions and successfully discover many passwords.

Another way in which administrators can set up password specifications is by banning certain sequences of characters. For example, Tufts does not allow users to repeat a character more than four times, while the University of Colorado disallows character repetition three times. These sequences are not secure because the may appear in some wordlists or fulfill simple regular expressions. UC Berkeley also has a unique guideline of having no more than five letters consecutively. This guideline may aid in preventing users from choosing a dictionary word and also requires that a required number or special character be included in the middle of the password, rather than just at the end. Also, though this may prevent crackers from successfully using certain regular expression rules (such as a string of letters preceded or followed by a number), it still allows character substitution.

The most popular ban in policy was passwords based off the username or some of the user's other personal information, such as full name, date of birth, or school. Eleven of the fifteen sources explicitly included this instruction, including all the academic institutions except for Pomona College and Scripps College. Even variations of the login name, such as spelling it backwards or rotating the letters, do not form good passwords

precisely because password crackers have the ability to perform these functions. On the other hand, the ban on personal information applies more to an adversary who knows the victim or has previously stolen information about the victim. Personally identifiable information that is commonly known directly violates the goal of a password: to be a secret that no one else knows or can easily guess. While systems can implement checks to guard against using one's username as the password, it is impossible to ban passwords that feature someone's pet's name or other likewise information simply because a computer or an algorithm does not know this information. At best, administrators can instill the importance of avoiding these passwords and hope that users follow their advice.

Given all the restrictions on passwords, how can a user create a good password? Four sources (Geodsoft, Harvey Mudd, Stanford, and Tufts) suggest using a mnemonic by thinking of a phrase, taking the first letter from each of the words, and replacing or inserting numbers or special characters. The advantage is that the phrase will be easy enough to remember, but the acronym will most likely not form a dictionary word, so wordlists will not be effective. Assuming the length of the password is sufficiently long, there is enough randomness in the acronyms that the probability of a pure brute-force attack will find the password is low. However, Narayanan and Shmatikov [9] note that "phonetic similiarity with words in the user's native language is a major contributor to memorability." A standard from natural language processing called the Markov model calculates the probability distribution of letters in a language. The authors state that a zero-order Markov model, in which each character is considered independently of the other characters in the password, is a good approximation for these acronym passwords.

Studying English only, they found that half of all plausible passwords can be found by looking at only 2.5% of the password space, and were able to recover 33 of 79 passwords that had length greater than 6. Examining such a small percentage of the total password space makes this sophisticated attack quite possible and reveals that the need for passwords to be human-memorable directly conflicts with the security of a password.

The period for which a password should be valid is still a matter of debate. For sites that require changing passwords, the average time that a password is valid is three to six months. The argument against required changes is that users already have too many passwords to memorize and, if forced to change their password, they will take insecure actions such as writing down their password or using the same password for multiple systems. Riley reports that people have an average of 8.5 accounts to maintain and 52.7% never change their password if not required [15]. Similarly, European internet security company NTA conducted a survey in 2002 and found that 30% of users have to remember over ten passwords [7]. In order to cope, 49% write their passwords down and 67% rarely or never change their passwords [7]. While these actions may be in violation of the password policy, there is no mechanism to ensure that the users follow the rules. The tradeoff between ease of use, namely memorability, and security is clearly problematic; stale passwords risk being cracked and enabling unauthorized access for long periods of time, but frequent forced changes create weak passwords that are more likely to be discovered through non-technological means. Since most users will not change their password until mandated to, a system administrator who decides that the benefits of changing passwords outweigh the risks must implement a mechanism to

enforce this policy. Otherwise, the expected behavior of the users will not match their actual actions, which is a risk in itself.

Passwords alone may not be sufficient to provide security as the smart dictionary attack shows, but systems are unlikely to move away from this model anytime soon. Therefore, policies must make passwords as strong as possible so that the more common techniques of attacking do not have as high a chance of succeeding. While each company or school may have specific needs, the flexibility of password cracking programs like John the Ripper create the need for strong and comprehensive password policies that can actually be enforced. Common elements of password policies today that a standard policy should include are a minimum password length of at least seven characters, a large input space of characters including lowercase and uppercase letter, numbers, and special characters, and the required exclusion of dictionary words and personal information such as login names. Other security mechanisms should be deployed in the near future in order to defend against the growing sophistication of password cracking methods, but at the very least, deploying and enforcing more comprehensive password policies now would provide more security than is often available today.

Works Cited

1. Andress, Amanda. *Surviving Security: How to Integrate People, Process, and Technology*. Boca Raton: Auerbach, 2004.

2. Anonymous. *Maximum Security*. USA: Sams, 2001.

3. Cliff, A. "Password Crackers: Ensuring the Security of Your Password." Security Focus. http://www.securityfocus.com/infocus/1192

4. Furnell, Steven. *Computer Insecurity: Risking the System*. London: Springer-Verlag, 2005.

5. Geodsoft. How-To: Good and Bad Passwords
http://geodsoft.com/howto/password/

6. Harvey Mudd College. Getting Started with Your Turing Account.
http://www.cs.hmc.edu/qref/starting.html#passwd

7. Hayday, Graham. "IT Users in Password Hell." ZDNet News.
http://news.zdnet.com/2100-1009_22-976888.html

8. MIT. Passwords. http://web.mit.edu/network/passwords.html

9. Narayanan, A. and Shmatikov, V. 2005. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security* (Alexandria, VA, USA, November 07 - 11, 2005). CCS '05. ACM Press, New York, NY, 364-372

10. Openwall wordlists collection for password recovery.
http://www.openwall.com/wordlists/

11. Openwall. John the Ripper: Rules.
http://www.openwall.com/john/doc/RULES.shtml

12. Penn State. Policies, Guidelines, and Laws: Password Policy
http://its.psu.edu/policies/password.html

13. Pinkas, Benny and Tomas Sander. Securing Passwords Against Dictionary Attacks. In *Proceedings of the ACM Computer and Security Conference (CCS' 02)*, p. 161-170. ACM Press, November 2002.

14. Pomona College. Password Expiration Policy.
http://www.its.pomona.edu/about/directorscorner/updates.htm

15. Riley, Shannon. *Password Security: What Users Know and What They Actually Do*.
http://psychology.wichita.edu/surl/usabilitynews/81/Passwords.htm

16. Shema, Mike and Bradley C. Johnson. *Anti-Hacker Tool Kit, Second Edition*. New York: McGraw-Hill/Osborne, 2004.

17. Scripps College. Change Password in Win XP.
`http://www.scrippscollege.edu/dept/it/guides/ChangePWinXP.htm`

18. Stanford University. Suggestions for Selecting Good Passwords.
`http://www2.slac.stanford.edu/computing/security/password/goodpassword.htm`

19. Tufts University. Creating Strong Passwords.
`http://www.tufts.edu/tccs/r-strongpass.html`

20. UC Berkeley. Passwords
`http://socrates.berkeley.edu/Acct/password.html`

21. UCLA BOL. Changing/Resetting BOL Passwords.
`http://www.bol.ucla.edu/services/accounts/info/password.html`

22. UCSD. INNOPAC System Password Use Policy.
`http://orpheus.ucsd.edu/systems/policy/innopac_password_use.html`