password protection for modern operating systems

The purpose of this paper is to help readers understand the security of the password encryption methods used in various operating systems and to establish some best practices for password management without requiring a background in cryptography. The article covers most pertinent background material in the first three sections.

by Steven Alexander

Steven is a programmer at Merced College. He programs for the Student System and manages the college's intrusion detection.

alexander.s@mccd.edu

Introduction

Early computer systems offered little in the way of password protection. The earliest designs stored a user's actual password along with his or her identifying information (username and/or user ID) in a central password file. Such schemes suffer from the obvious problem that any user who either legitimately has or surreptitiously gains access to the password file knows the password to every account on the system. Later designs avoided this problem by storing only an encrypted or hashed value in the password database.

Such systems are still vulnerable. Any attacker with access to a list of encrypted passwords can guess passwords by encrypting words from the dictionary or at random and comparing the encrypted results of his or her guesses against the encrypted passwords in the database. The designers of the UNIX operating system improved on this method by using a random value called a "salt." A salt value ensures that the same password will encrypt differently when used by different users. This method offers the advantage that an attacker must encrypt the same word multiple times (once for each salt or user) in order to mount a successful password-guessing attack.

Cryptography: The Basics

Encryption preserves the confidentiality of data. A user encrypts a message using a secret key so that others who do not know the secret key cannot recover the message. The message is called plaintext before encryption and ciphertext after encryption. Password protection schemes also use encryption, typically as a secure hash function.

Many modern encryption algorithms are iterated block ciphers that break data into blocks of a specified size and encrypt them by iterating an encryption function several times; each of these iterations is a round. One such cipher, Blowfish, is discussed later.

Rather than use the whole key to perform encryption in each round, most iterated block ciphers derive a subkey for each round using a "key schedule." A simple key schedule may select certain bits of the key for use in each round, whereas a more complicated key schedule, such as in RC5, may use mathematical means to generate the subkeys from the key.[1]

A hash function takes an input and applies a set of operations to reduce the input to a numeric value of a fixed size (usually 128, 192, or 256 bits). Outside of cryptography, programmers use non-secure hash functions to sort and search data. Secure hash functions need to have the special properties that it is computationally infeasible to determine a message from its hash value and that it is difficult to find another message with the same hash value.

1. Alfred Menezes et al., *Handbook of Applied Cryptography.* CRC Press, 1997.

Because it is difficult to find two messages with the same hash value, hash values are used to verify the integrity of messages. Users can independently compute the hash value of a message and compare it to a known source to verify that the message has not changed. Operating systems often apply a hash function to a password and store the encrypted result instead of the plaintext password. Because of the properties described, it is hard to determine the original password from the hash value and it is difficult to determine another password that has the same hash value (though one or more might exist). A user who needs to authenticate to the operating system tells the system his or her password, and the system hashes it and then compares the resulting value with the value stored in the password hash database.

Some operating systems use a dedicated hash function such as MD4 or MD5 to protect passwords; others use an encryption algorithm such as DES or Blowfish. When an encryption algorithm is used, the system uses each password as the secret key to encrypt a known message. In this case, the message is not secret, the systems designers only wish to prevent an attacker from guessing the key used to encrypt a message. The reader of this article should be aware that what is often referred to as password encryption is really password hashing. To add to the confusion, encryption algorithms are sometimes used as hash functions (albeit with some modification in use or design).

Early UNIX

The standard password hashing algorithm in UNIX, crypt, is a modification of the DES encryption algorithm;[2] it is often referred to by its man page entry, crypt(3). The system hashes each password after combining it with a 12-bit (4096 possible combinations) salt value. UNIX uses the salt value to modify one of the properties of the DES algorithm; this means that the same password will hash to a different result after merging each with a different salt value.

The salt value is not secret; it is stored with the hashed password. When a user presents a password to UNIX for authentication, the operating system looks up his or her salt value and hashed password. UNIX uses the user's stored salt value to modify the DES algorithm when it hashes the presented password; the system then compares the result with the value stored in the password database. Changing the DES algorithm with a salt value has the added benefit that an attacker cannot use off-the-shelf DES encryption hardware to brute force passwords.

Whenever a user chooses a new password, the system generates a new salt at random. Historically, UNIX generates the salt from the time of day or some other weak source of entropy (randomness). Because of this, it is common for two users at a site to have the same salt value. Still, the complexity of an offline password-cracking attack greatly increases because each word must be re-hashed for each salt value.

The designers of UNIX also introduced another important idea in the design of the UNIX crypt algorithm: They increased the time necessary to create or verify a password in order to further thwart offline password-cracking attacks while still keeping system response to an acceptable level. UNIX iterates the DES algorithm 25 times to create the crypt algorithm. Provos and Mazières estimated that a Digital Equipment Corporation VAX-11/780 contemporary to the design of crypt would be able to try only 3.6 possible passwords per second per salt.[3] If a system had 36 users, each with a different salt, it would take 10 seconds to try each possible password. Of course, modern equipment is able to fare much better; improved algorithms have replaced crypt in several modern UNIX variants.

2. Menezes et al., *Handbook*; Robert Morris and Ken Thompson, "Password Security: A Case History," *Communications of the ACM*, vol. 22, no. 11 (November 1979), pp. 594–97.

3. Niels Provos and David Mazières, "A Future-Adaptable Password Scheme," *Proceedings of the 1999 USENIX Annual Technical Conference* (June 1999), *http://www.usenix.org/events/usenix99/provos.html*.

Recently, researchers at the San Diego Supercomputer Center instituted a project to store the hashes of over 50 million common passwords computed once with each of the 4096 possible hashes used by the DES crypt mechanism.[4] Their results would be less worrisome if they had not shown that the requirements for such an attack are also within reach of a group of attackers using distributed storage.

Windows NT/2000/XP

The Windows NT line of Microsoft operating systems stores two password hashes: the LanMan hash and the NT hash.[5]

LanMan

The LanMan hash is used for backwards compatibility with Windows 95/98 and is the less secure of the two Windows hashes. Windows limits passwords to 14 characters for the LanMan hash. The system computes the LanMan hash by splitting the password into two seven-character halves and converting all characters to uppercase; if the password is less than 14 characters, the system pads it with null bytes. The system uses each half as a secret DES key to encrypt a fixed string of plaintext. The resulting hashes are concatenated.

Programs such as L0phtCrack have exploited the weaknesses in the LanMan hash. LanMan passwords are not case-sensitive. In addition, the system treats a 14-character password as two seven-character passwords. To illustrate the problem, let us consider the possible number of combinations for passwords of a given length and character set.

If x is the size of the character set used for a group of passwords (for instance, 26 if the passwords are alphabetic and not case-sensitive), then there are x^y possible passwords of length y with that character set. So a seven-character password that contains only numbers and uppercase letters has $36^7 = 78{,}364{,}164{,}096$ possible values.

An alphanumeric 14-character password should have $36^{14} = 78{,}364{,}164{,}096 \times 78{,}364{,}164{,}096$ possible values. Because Windows treats all passwords as two separate seven-character passwords for the purpose of computing the LanMan hash, an attacker only has to try $2 \times 36^7 = 2 \times 78{,}364{,}164{,}096$ possible values. If we assume case insensitivity, it is approximately 39 billion times easier to break two alphanumeric seven-character passwords than it is to break an alphanumeric 14-character password. If you had a computer that was capable of breaking any alphanumeric seven-character password in one second (quite a feat!), you would have to wait up to 1200 years to break an alphanumeric 14-character password.

Case sensitivity is a problem. A seven-character alphanumeric LanMan password has $36^7 = 78{,}364{,}164{,}096$ possible values, since Windows converts all letters to uppercase before it computes the hash. However, if the algorithm were case-sensitive, there would be 62^7 possible values, a total of 3,521,614,606,208. There are 45 times more seven-character case-sensitive alphanumeric passwords than seven-character case-insensitive alphanumeric passwords.

NT Hash

Windows computes NT hashes by applying the MD4 hash algorithm, invented by Ron Rivest, to the entire password; there is a 14-character maximum on Windows NT but not on Windows 2000 or XP. The NT hash does not suffer from the same problems as the LanMan hash. The NT dialect hash is case-sensitive and computed against the

4. Tom Perrine, "The End of crypt() Passwords Please?" *;login:*, vol. 28, no. 6 (December 2003), pp. 6-12.

5. Bruce Schneier and Mudge, "Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP)," *Proceedings of the 5th ACM Conference on Communications and Computer Security* (November 1998), *http://www.schneier.com/paper-pptp.html*.

entire password. Administrators who do not need to support Windows 95, 98, or Millennium Edition users are encouraged to disable the storage of the weaker LanMan password.[6]

6. SANS, "SANS Top 20 Vulnerabilities," *http://www.sans.org/top20/*.

The MD4 algorithm consists of 48 steps which turn a 512-bit input into a 128-bit output. MD4 pads all inputs to a multiple of 512 bits, though it can iterate over several 512-bit blocks if necessary. When passwords are 13 characters long or less, the inputs to the last three steps are null. An attacker can use this information to reverse the last three steps of any password hash he or she is trying to crack. Subsequently, the attacker only needs to compute the first 45 steps for each password tried. This results in a speedup of about 6%.

Further improvements to this optimization are possible. The 128-bit output of MD4 is really four separate 32-bit values. Only one of these values changes in each step. The fourth of these 32-bit values changes last in steps 41 and 45. Since step 45 is reversed by the previous optimization, an attacker can compare the fourth part of the hash value after step 41 to the calculated value and stop computing if the values do not match. An attacker will only have to compute beyond step 41 once in every 4 billion tries. This improves the speedup by about 15%.

Common Problems

The Windows password schemes suffer from some common problems. First, there is no salt value applied to the passwords. Because of this, attackers can hash a large dictionary of possible passwords in advance to speed up their attacks. Using efficient sorting and searching methods, it is a trivial matter to determine whether a user's password hash corresponds to one of the hashes in the attacker's dictionary. An attacker who is actively guessing passwords against a large list of users can save an enormous amount of computation time by only having to encrypt each possible password once, unlike the case with UNIX passwords.

Sort-and-search algorithms make an attacker's job even easier. The naive method for finding a Windows password is to hash a possible password, then compare the result against the password hash of every user whose password you wish to recover. This is akin to reading every entry in the phone book until you find the person you're looking for. By sorting the password hashes, an attacker only needs to compare the hash of each possible password against a small number of password hashes.

UNIX crypt()	249467 hashes/second
BSDI DES (x725)	9618 hashes/second
FreeBSD MD5	4452 hashes/second
OpenBSD Blowfish	335 hashes/second
Kerberos AFS DES (short)	244907 hashes/second
Kerberos AFS DES (long)	435745 hashes/second
Windows NT LanMan DES	628234 hashes/second

Table 1

Another drawback to the password hashing schemes used in Windows is, unfortunately, efficiency. Faster password hashing algorithms allow an attacker to guess passwords more quickly. Table 1 offers a comparison of the speed of different password hashing algorithms as implemented in the popular password cracker "John the Ripper." The benchmarks in Table 1 are from the program's own benchmarking feature as it performed on a 2.4GHz Pentium 4 with 512MB RAM. I have noticed that John performs about 40–50% faster in practice (on my reference machine) than the benchmarks show; this is possibly a cache issue. L0phtcrack 4 performs significantly better than John and is able to compute about 5.3 million LanMan hashes per second; it computes about 126,000 NT hashes per second. The difference is very likely rooted in machine-specific optimizations, so your mileage may vary.

Obviously, it would do no good to slow down a password hashing routine to the point that it bottlenecks a system; however, other modern operating systems have slowed the process of password hashing in order to hinder would-be attackers. The key here is to balance security and efficiency.

New and Improved Problems

Recently, Philippe Oechslin, improving on a technique developed by Martin Hellman in 1980, developed a cryptanalytic attack that has consequences for Windows passwords.[7] Readers interested in the details of the attack should read Oechslin's paper. This article will only cover the attack in a basic Windows-specific manner.

The attack is a time-memory tradeoff that requires an attacker to store up to several gigabytes of data. If storage were of no consequence, an attacker could pre-compute the hashes of every imaginable password up to some reasonable length and store them along with the accompanying password. Attackers would then only need to look up a given hash in their database to discover the password. Oechslin's attack is able to achieve nearly the same convenience with thousands of times less storage.

The attack uses large tables of "rainbow chains." They are computed as follows:

1. A random word is generated and stored in the first column of the current row in the table.
2. The word is encrypted and the resulting ciphertext is "reduced" so that it represents another possible password (each character is converted into a printable ASCII character).
3. Step 2 is repeated many times over (usually a few thousand). The length of the chain dictates the number of times that step 2 is repeated. A part of Oechslin's improvement on Hellman's method is to slightly change how the reduction is computed after each repetition in the chain. The details and consequences require a paper of their own.[8]
4. The result of the final reduced value is stored in the second column of the current row.

Once an attacker generates sufficiently large tables, he or she will be able to recover a significant number of passwords in a small amount of time. An attacker first tries to determine if the password used to compute a given hash is the same as one used to compute one of the chains in his or her tables. This is determined by reducing and hashing the password hash in a manner similar to the method listed above for creating the tables. Each time the password hash is reduced, it is compared to the entries in column two of the table. If a match is found, the chain is recomputed (beginning with the value in column 1). The attacker will have to weed through some false-positives to find the target password. If one of the passwords hashed during the creation of the chain produces the password currently under attack, the attacker is successful. It takes several seconds to find a password using typical current workstations. The search takes longer (and fails) if the password is not a part of any of the stored chains.

This attack is statistical in nature. For alphanumeric LanMan passwords, a 2.8 gigabyte table will include about 99.9% of the possible passwords. An attacker actually has to perform about 10 times as much computation to generate such a table of rainbow chains as would be needed to compute and store every possible password and hash. This is not as bad for the attacker as it seems. Using a 2.4GHz Pentium IV with 512 megabytes of RAM, I was able to generate five rainbow tables, each with 35 million

7. Philippe Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off," *Crypto* 2003 (forthcoming). Douglas Stinson, *Cryptography: Theory and Practice.* CRC Press, 1995.

8. Oechslin, "Making a Faster Cryptanalytic."

chains of length 4666, in about 215 hours (nine days). These five tables are equivalent to the single 2.8 gigabyte table mentioned above.

Oechslin applied his technique to the Windows LanMan password hash; the attack is applicable to the Windows NT Hash with more effort. The attack is more difficult to use against the NT Hash because it is case-sensitive and the passwords are not limited to seven characters in length. Attackers could, of course, simplify their attacks by only considering passwords of seven characters or fewer. Still, an attacker would have to consider that there are a far greater number of possible case-sensitive passwords than case-insensitive ones. The attacker would probably consider a lesser set of passwords, such as those where all characters happen to be lowercase or where only the first letter is in uppercase.

FreeBSD

By default, the FreeBSD operating system uses a crypt mechanism based on the MD5 hash algorithm. MD5 is the successor to the MD4 algorithm used for password hashing in the Windows NT line of operating systems.

Poul-Henning Kamp developed the MD5 crypt routine based on Rivest's MD5 hash algorithm. MD5 crypt uses a salt of up to 48 bits and effectively has no limitation on password length. It is also far slower than either DES crypt or the Windows password hashing methods. To achieve this, MD5 crypt uses an inner loop with 1,000 iterations to continuously remix data into the hash calculation. FreeBSD also supports the traditional DES-based crypt and a Blowfish-based crypt mechanism. FreeBSD distinguishes MD5 and Blowfish hashes from DES crypt hashes by adding a prefix to the hash entries.

Provos and Mazières raised questions about the design of MD5 crypt;[9] however, the algorithm currently looks to be far more secure than the DES crypt mechanism or either of the Windows password hashing schemes. There is a better alternative to all of these.

OpenBSD

Niels Provos and David Mazières have designed a crypt mechanism based on the Blowfish encryption algorithm.[10] Blowfish is a block cipher encryption algorithm designed by Bruce Schneier.[11] Provos and Mazières actually designed two algorithms, eksblowfish and bcrypt. Eksblowfish is derived from Blowfish and has a purposefully slow key schedule. Bcrypt is a hash algorithm based on eksblowfish. Bcrypt is the most secure password hashing algorithm in common use at the time of this writing.

Bcrypt allows passwords to be up to 55 characters in length. Note that while MD5 allows longer passwords than bcrypt, this does not increase its security, because its 128-bit output is the limiting factor. It would be easier to find an alternate password with the same hash as a given password than to find a specific password in excess of 55 characters. A random password consisting of only printable ASCII characters only needs to be 20 characters long before a hash function output of 128 bits is the limiting factor. A hash function with a 192-bit output limits the security of passwords of 30 characters or more. Passwords that are not completely random would need to be longer to provide the same security; however, the limit of current supercomputing technology is close to 70 bits and will not approach 128 or 192 bits anytime soon.

Bcrypt requires a random salt value of 128 bits, which is large enough that no two accounts on the same system are ever likely to have the same salt. In fact, an attacker

9. Provos and Mazières, "A Future-Adaptable Password Scheme."

10. Provos and Mazières, "A Future-Adaptable Password Scheme."

11. Bruce Schneier, "Description of a New Variable Length Key, 64-Bit Block Cipher (Blowfish)," Fast Software Encryption, *Cambridge Security Workshop Proceedings.* Springer Verlag, December 1993, pp. 191–204.

would need to have the hashes of about 16 quadrillion users before it is more likely than not that two hashes are alike.

Bcrypt also uses a cost variable; an increase in the cost variable causes a likewise increase in the time required to perform a bcrypt hash. The cost assigned to new passwords is configurable using a systemwide configuration file. In OpenBSD, administrators can assign different cost values for normal users and the superuser.

Protecting Password Hashes

Storage

Password hashes need protection regardless of the security of the hashing mechanism. An attacker lacking the password hashes for a system cannot attempt any offline attacks.

Most UNIX systems offer password shadowing. When password shadowing is used, user information is stored in the /etc/passwd file but the password hashes are stored in another file, usually /etc/shadow or /etc/master.passwd. Many UNIX systems automatically use password shadowing; others (HP-UX for instance) require an administrator to configure password shadowing. All system administrators are encouraged to familiarize themselves with the pertinent areas of their system documentation.

When possible, UNIX administrators should use MD5 crypt or Blowfish instead of the traditional DES crypt. Both of these alternatives are available on Linux, Solaris, and the BSD systems. For information about Blowfish on Linux, please visit OpenWall (*http://www.openwall.com/crypt/*). Sun Microsystems recently introduced their own crypt mechanism based on MD5. The new mechanism is meant as a more secure replacement for the MD5 crypt mechanism introduced for FreeBSD. Sun's new algorithm uses a configurable number of iterations for its inner loop. The default value is currently 4096. I do not currently know if the inner loop is the same as FreeBSD's but, with the high number of iterations, it looks to be much slower (which is good!). To my knowledge, these alternative crypt routines are not currently available on AIX, IRIX, or HP-UX. Replacing DES crypt may break some upper-level applications, particularly those run from UNIX operating systems that do not support the new methods; consider yourself warned.

Microsoft introduced SysKey, with the release of Windows NT Service Pack 3, to encrypt password hashes stored in the Windows registry. Attackers can bypass SysKey protection using pwdump2. For more information on SysKey use, Windows administrators should refer to Microsoft's "knowledge base" articles.[12]

Transmission

Sending unencrypted passwords across a network is an activity best reserved for those who like to live dangerously. Many administrators erroneously think that switched networks will prevent an attacker from sniffing passwords as they travel across a network; this is not true.[13] Switches enter a "learning" mode after they start up. While in this learning mode, a switch will broadcast traffic in the same manner as a hub. The MAC tables on a switch can also be selectively poisoned, which allows traffic interception, as is done by programs like Dsniff and Ettercap, or they can be overloaded so that legitimate entries are flushed from the table, which will force the switch to broadcast traffic destined for those addresses.

12. Microsoft, "Windows NT System Key Permits Strong Encryption of the SAM," Microsoft Knowledge Base Article 143475; Microsoft, "How to Use the SysKey Utility to Secure the Windows 2000 Security Accounts Manager Database," Microsoft Knowledge Base Article 310105, *http://support.microsoft.com/*.

13. Abe Singer, "No Plaintext Passwords," *;login:*, vol. 26, no. 7 (November 2001), pp. 83–91.

The security personnel at the San Diego Supercomputer Center have eliminated the transmission of unencrypted passwords on their protected network using solutions such as SSH and SSL.[14] System administrators are encouraged to read Abe Singer's paper and consider what they can do for their own networks.

Some protocols that enable the elimination of plaintext password transmission have other drawbacks. For instance, the Windows NTLM protocols use a challenge response mechanism.[15] In the NTLM protocols, a server sends a random challenge to a client that has requested authentication. The client encrypts the challenge using a user's password hash and sends it back to the server. The server attempts to decrypt the client's response using the copy of the user's password hash stored on the server.

This scheme suffers from the obvious problem that a user only needs the password hash to authenticate; an attacker able to recover the hashes from the server can operate just as well as if he or she had the actual passwords. This version of the protocol has numerous other problems as well. Version 2 of the NTLM protocol has better security properties than its predecessor; administrators are encouraged to upgrade.[16] The Kerberos protocol also suffers from the problem that all of the information needed to authenticate is stored on the server.[17]

Password Policy

Password Handling

Often, the users of a system will bypass all of the carefully designed and maintained security mechanisms of that system to convenience themselves. It doesn't matter how securely passwords are chosen or stored in a system if users have those passwords stuck to their monitor or keyboard on a sticky note. There are two things to consider here: policy needs to strictly forbid such activity, and users need to have passwords that they can remember. Unfortunately, effectively enforcing policy requires the involvement of management. Getting all of the management in an organization to enforce rules against post-it notes may require divine intervention. However, other approaches may have better success.

One approach is for system policy to specifically authorize IT staff to confiscate post-it notes with password information and disable the related account. This should eliminate visible notes but probably won't prevent users from sticking notes under their keyboard or in the top drawer of their desk. User education can help increase adherence to policy.

Writing a password down is not always bad. It is quite reasonable to have system administrative passwords written down in a secure location. Of course, proper procedures must be maintained for handling the passwords. Sometimes, an administrator may need to keep a written copy of passwords temporarily. Consider, for instance, the case that a successful intrusion has happened and the administrator must change several different passwords at once. Some administrators solve this problem by choosing multiple passwords that are variations of each other. This method suffers from the problem that password cracking software might generate those same variations if one of the passwords is cracked successfully and added to an attacker's dictionary. If administrators (or users) must write down password(s), I recommend that they are stored in a semi-secure location, such as a locked cabinet or drawer (in a non-public area) or a wallet (if kept on the administrator's person). Change the passwords immediately if the written copy is lost.

14. Singer, "No Plaintext Passwords."

15. Hobbit, "CIFS: Common Insecurities Fail Scrutiny" (January 1997), *http://www.insecure.org/stf/cifs.txt*; Schneier and Mudge, "Cryptanalysis of Microsoft's PPTP"; Bruce Schneier et al., "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)," *CQRE '99* (Springer Verlag, 1999), pp. 192–203, *http://www.schneier.com/paper-pptpv2.html*.

16. Schneier et al., "Cryptanalysis of Microsoft's PPTP Authentication Extensions"; Microsoft, "How to Enable NTLM 2 Authentication," Microsoft Knowledge Base Article 239869 (2004), *http://support.microsoft.com/*.

17. Rik Farrow, "Network Defense: Kerberos for Net Authentication," *http://www.spirit.com/Network/net0902.html*; Simson Garfinkel and Gene Spafford, P*ractical UNIX and Internet Security*. O'Reilly, 1996.

Many organizations have periodic staff development activities; if management can be convinced to sponsor a workshop that includes basic security information and the reasoning behind policy, users may adhere to it more closely. It may also be effective for IT staff to periodically share information about attacks on the network. I am not suggesting that anybody should try to convince their system's users of an impending electronic doomsday, only that some awareness is helpful.

Another problem in most organizations is password sharing. Eliminating this problem involves both technical and policy measures. Restricting users to a single logon session can help to alleviate the problem. If possible, policy should authorize the IT department to disable accounts that it knows a user has shared. It is also important that it be as simple as possible for users to gain access to objects to which a user should legitimately have access. If it takes days for a user to get access to the resources that they need, they are much more likely to try to share another's account. In addition, in such an environment, users are more likely to be sympathetic to other users and share their password when asked.

PASSWORD SELECTION

The weakest passwords are short or are words, names, or derivations of words or names. Simply changing an "A" to a "4" or an "s" to a "$" does not significantly increase the security of a password. The password cracker John the Ripper has a customizable set of rules that it uses to try various permutations on supplied dictionary words and user information as possible passwords.

The three properties that define the security of a password are: length, character set, and randomization. Strength in one of these properties can make up for weaknesses in the others (to a point).

When passwords are not limited to seven or eight characters, as in Windows NT or some UNIX systems, length is the easiest way to increase the security of a password. It is easier to choose a very long password than a very random password. A 40-character password that uses only lowercase letters and spaces will be extremely difficult to break. Using current computing technology, such a password would be impossible to break without good language analysis or luck. Still, it is not wise to pick passwords from a popular text such as one of Shakespeare's plays or sonnets.

The characters used in a password have a major effect on the security of the password, especially when the length of a password is limited. An eight-character password with lowercase letters and punctuation is over 800 times harder to break than a password of the same length with just lowercase letters. When considering the characters used in a password it is useful to break the character set into groups: uppercase and lowercase letters each account for 26 characters, numbers account for only 10, and special characters account for 34. Strong passwords should have at least one character from each of three different groups.

Truly random passwords are hard to remember. It usually suffices to generate a password from a pattern that is meaningful only to you. "TfcIdwaT" is meaningful only if you know that it stands for "The first car I drove was a Toyota." Adding or prepending a number to this password would make it even better (if it wasn't in print).

On Microsoft Windows 2000 and later, there is an option in the Local Security Policy, accessed through Administrative Tools in the Control Panel, that "Password must meet complexity requirements," which can be enabled to force users to use strong pass-

The passwords used for Windows accounts should not match those used for UNIX accounts.

words. It requires that users not base passwords on their username (in whole or in part), that passwords be at least six characters long, and that the password contain characters from at least three of the four character groups mentioned above. I think that this forms the basis for a good password selection policy on any system. My only complaint is that the length requirement is too low. Thankfully, Windows also allows a minimum password length to be set; I recommend 14 characters if storage of the LanMan password has not been disabled; otherwise, depending on the security requirements of the system, the value can be set as low as 10.

Various methods exist in other operating systems for enforcing password policy. OpenWall has made a PAM module available that allows flexible policy configuration (*http://www.openwall.com/passwdqc/*). Among other things, the module can modify password length requirements based on the character groups present in a given password. Many systems allow an administrator to configure a minimum password length even if PAM is not available. Recall that only the first eight characters matter if the traditional DES crypt is used.

Password Aging

Password aging is configurable on most systems and is important for a good security policy. I don't recommend expiring passwords more often than every 30 days; users are more liable to forget their passwords, reuse passwords, or write them down. Passwords for accounts with administrative privileges should expire every 30 to 90 days, whereas it may be acceptable to force the expiration of user passwords as infrequently as every 120 to 180 days. If possible, use password history to prevent users from reusing old passwords. My recommendation, if the option is configurable, is to remember one to two years' worth of passwords.

Administrators must consider several factors when deciding how often to change passwords. If the security requirements of the network are high, passwords should expire more frequently. If the password hashing mechanism is weak, DES crypt, or a Windows method, the passwords should again expire more frequently. If an attacker gains administrative access, expire all passwords immediately (aside from other measures). If passwords are not transmitted across the network in the clear, passwords can be changed less frequently. Systems that force users to choose strong passwords of 10 characters or more, except systems using LanMan or DES crypt, can afford to allow passwords to expire a little less often.

It is important to remember that, regardless of their cryptographic strength, passwords can be captured by sniffing, keystroke logging, or other methods. A major point to consider is the willingness of your user population to comply with the requirements and not break security by other means (such as sticky notes).

Passwords on Multiple Systems

To whatever extent possible, administrators and users should use different passwords for different systems. The passwords used for personal accounts should never be the same as those used in the workplace. I would also recommend that passwords be different for different classes of systems in the workplace.

To clarify this statement the passwords used for Windows accounts should not match those used for UNIX accounts. One can consider network devices such as routers and switches a single class, but they should be separate from security-critical devices such as intrusion detection systems and firewalls.

The key is that there should be some separation. If passwords are the same or similar across multiple systems, an attacker is more likely to leverage access on one machine to gain access to the rest of the network. This is especially true of passwords used to access third-party server software; several years back I discovered that the password encryption used in a popular email server for Windows was just a simple substitution cipher. Six months later, a security company discovered that the company had replaced the password encryption algorithm with another equally simple cipher. Be careful and use your own judgment when reusing passwords.

Acknowledgments

I extend my thanks to Keith Simonsen for his suggestions and Casper Dik for answering my last-minute questions.

Save the Date!

OSDI '04

Sixth Symposium on
Operating Systems Design
and Implementation

December 6–8, 2004 ◆ San Francisco, CA

Co-located with WORLDS '04

http://www.usenix.org/osdi04/