CHAPTER 10

Challenge Response Passwords

President Skroob: 1-2-3-4-5? That's amazing! I've got the same combination on my luggage!

— Mel Brooks et al., *Spaceballs*

IN THIS CHAPTER

This chapter examines a more general mechanism for one-time password authentication in which the person must respond to a mathematical challenge by the authentication mechanism. The response is mechanized in various ways.

- Evolution of challenge response authentication
- Challenge response with X9.9 and S/Key
- Microsoft LAN Manager authentication
- Microsoft Windows NT LAN Manager authentication

10.1 Challenge Response

In 1979, the "personal computer revolution" was in full swing, with new products vying for hobbyists' attention. Although many computers appealed to hardware hackers and arrived in kit form, a newer breed appealed to software hackers. These computers, like the Apple II and the TRS-80, arrived prebuilt and ready to plug in, with some extremely simple but usable software development tools. The TRS-80, built by Tandy and sold through its Radio Shack stores (hence the "TRS"), caught the attention of engineer Bob Bosen.

In his spare time, Bosen wrote a game for the TRS-80 called *80 Space Raiders*, and advertised it for sale in hobbyist magazines. At that time, many hobbyists considered software piracy a routine activity, or even a moral imperative, so Bosen embedded a copy protection mechanism in his game. As it was, the income from game sales had barely paid its advertising bills, even though the game had received positive reviews and satisfied reactions from customers. ***see Note 1.***

Bosen's copy protection consisted of a *challenge response* mechanism. When someone started the game running, it displayed a randomly selected number, called the *challenge*. The game's owner had to look for that number in a table supplied with the game, and type in the corresponding *response*. The game would check the response to verify that it went along with that particular challenge. Thus, one needed a copy of the table in order to play the game. Bosen also took steps to make the table hard to copy: he printed it on red paper and attached to the side of the curved tube that served as the "box" for 80 Space Raiders.

The game wasn't a real commercial success, but several people suggested that a variant of the copy protection scheme could authenticate people to timesharing systems and mainframes. Bosen took the idea and started Enigma Logic, Inc., to produce SafeWord challenge response authentication tokens. Bosen also filed for patents on challenge response tokens in 1982–83. A British patent was issued in 1986, but the U.S. patent application encountered lengthy, unexplained delays and was never issued.

Challenge response authentication still works essentially the same as 80 Space Raiders copy protection: the system displays a number, called a *challenge* or *nonce*, and the person must provide the corresponding response. These days, however, most systems use a one-way function to compute the response instead of embedding the responses in a printed table. People often generate responses to authentication challenges by computing the one-way function on a specially built calculator: the password token. There are several commercial tokens that support challenge response authentication, including SafeWord, WatchWord, ActivCard, and CryptoCard.

People sometimes refer to tokens that use challenge response as *asynchronous* tokens, when compared to the *synchronous* tokens described in Chapter 9. We could say that synchronous tokens are also challenge response tokens, except that the challenge is predictable and is generated automatically by the token and the server. What makes them synchronous is that the server and token must remain synchronized with respect to the challenges they generate.

More recently, many people use client software on their workstations to automatically handle challenge response authentication. Many vendors provide software versions of their authentication tokens, and challenge response is also used by Microsoft for its network server authentication, as described later in this chapter.

Figure 10.1 illustrates a typical challenge response system. In the example, Cathy needs to log on to a server using a challenge response system. She starts by telling the system her user name ("croe"). The server generates a nonce and sends it to Cathy as a challenge ("493076"). Cathy types the challenge into her one-time password token, and the token generates the correct password to send back in response ("319274"). The server looks up Cathy's base secret and uses it to compute the response that should match the challenge. If the server's computed result matches Cathy's response, then it lets Cathy log on.

A major advantage of the challenge response mechanism is that the servers do not need to maintain synchronization with people's tokens. By generating a random challenge each time, the server ensures that previously valid passwords are unlikely to be valid for the next login. No timers or counters are needed. On the other hand, users must key in additional numbers when using the challenge response token.

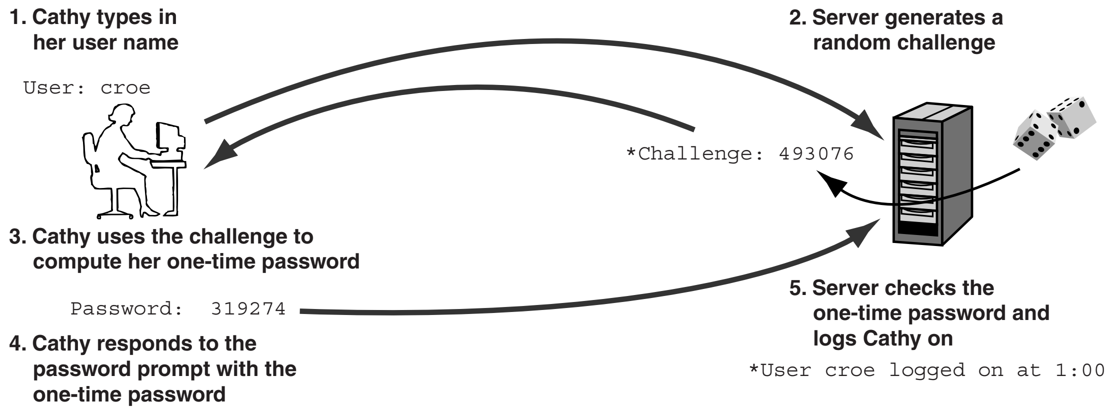
1. Cathy types in her user name
User: croe
2. Server generates a random challenge
*Challenge: 493076
3. Cathy uses the challenge to compute her one-time password
Password: 319274
4. Cathy responds to the password prompt with the one-time password
5. Server checks the one-time password and logs Cathy on
*User croe logged on at 1:00

FIGURE 10.1: *Generating a challenge response password.* The server sends a random number to Cathy. She uses her one-time password token to compute the correct response to that challenge. The correct response depends on the value of her base secret. If she has the right base secret, her response will match the response expected by the server. Then the server will log her on.

CHALLENGE RESPONSE AND X9.9

Unlike clock- and counter-based systems, challenge response is an "open standard" authentication mechanism. This makes it easier for the security community to examine these products and make more accurate judgments regarding the security they provide. Furthermore, open standards provide a mechanism by which products from different vendors can interoperate.

The U.S. government and the American Bankers Association have adopted a consistent standard for data authentication called FIPS 113 and X9.9, respectively. These standards used DES to serve the purpose of a one-way hash function. Figure 10.2 illustrates a form of this authentication in which the one-time password response "authenticates" the random challenge. Most challenge response tokens, including SafeWord, ActivCard, CryptoCard, and WatchWord, provide at least one mode that is X9.9 compatible. ***see Note 2.***

The password generation process shown in Figure 10.2 is very similar to the synchronous processes described in Section 9.3. Internally, challenge response tokens operate in the same general manner as clock- or counter-based tokens. Both use the base secret plus some varying data to generate the one-time password. Instead of using the clock or counter as the varying data, the challenge response token uses the challenge. X9.9 works as shown in the figure: the Data Encryption Standard (DES) takes the challenge as input and encrypts it using the base secret as the key. The challenge is padded with zeroes to make up the full 64-bit block of data that DES requires. Most vendors truncate the response to fit the

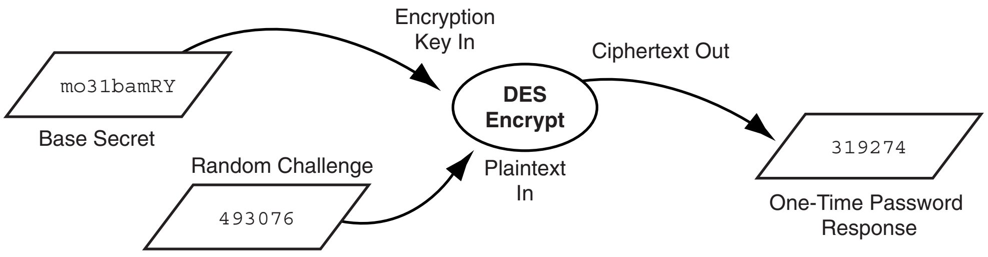

FIGURE 10.2: *Challenge response authentication.* The token stores the base secret internally. The owner types in the challenge. The token computes the one-time password response by encrypting the challenge using the base secret as the encryption key. The server uses the same process to verify the response.

TABLE 10.1: *Password Tokens and Average Attack Spaces*

Example	Style of Attack	Average Attack Space
Biometric with a 1% FAR (1 in 100)	Interactive	6 bits
Four-digit PIN	Interactive	13 bits
Biometric with a FAR of 1 in 100,000	Interactive	16 bits
ANSI X9.9 token with six-digit response	Interactive	19 bits
One-time password token	Interactive	19 bits
Eight-character personal password	Off-line	22.7 bits
ANSI X9.9 token using 56-bit DES	Off-line	54 bits
SecurID one-time password token	Off-line	63 bits

hardware token's display or to conform with a site-specific choice of the length of responses.

Table 10.1 compares the average attack space of ANSI X9.9 challenge response authentication with other authentication methods discussed in earlier chapters. The "interactive" attacks are those that must directly interact with an authentication mechanism; this usually slows the attack down and provides a way to detect the attack. The "off-line" attacks are computer-driven trial-and-error attacks that can't be detected by the targets of the attack.

A security advantage of challenge response passwords is that each password is associated with a particular attempt by the owner to log on. If the attempt fails for any reason (for example, the connection is lost), then the authentication process starts over with a new challenge. This mitigates the risk of an attacker's intercepting and reusing a one-time password (Attack A-56 in Section 9.4).

S/KEY AUTHENTICATION

S/Key is a one-time password system developed at Bellcore in the early 1990s as a technique for logging on to Unix systems. The technical concept was first proposed by Leslie Lamport and published in 1981. Unlike other challenge response techniques, Lamport's approach does not maintain a database of secret keys, so attackers cannot compromise the system by stealing the database of base secrets. ***see Note 3.***

Instead, Lamport's scheme uses a sequence of one-way hash values that are computed from a memorized password (Figure 10.3). As with more traditional Unix passwords, the system takes advantage of the fact that it's easy to compute the hash of a password, but impractical to derive a password from its hash. Lamport's scheme uses a sequence of hashes, each computed from the previous one in the sequence. The server stores the last hash in the sequence. To log on, Cathy provides the next-to-last hash in the sequence as a one-time password. The server takes her one-time password, hashes it, and compares it to the stored hash. Both should match. Then, the server replaces the hash in Cathy's password entry (the *fourth* hash) with the password she just provided (the *third* hash). The next time Cathy logs on, she provides her *second* hash and the server follows the same process.

S/Key is a practical implementation of Lamport's scheme. Strictly speaking, the technique uses synchronous one-time passwords rather than challenge response passwords. After all, the user always needs to provide the hash-before-last as the one-time password. This requires the user to keep a perfect count of which passwords have been used, and in practice, users aren't good at that sort of bookkeeping. Instead, S/Key servers generally prompt the user with the sequence number of the next hash expected. This makes S/Key act like a challenge response system, though the information is really optional and is only provided for the user's convenience.

The S/Key hash also incorporates a random value called a *seed*, which is combined with the base secret when generating the hashes. The seed prevents S/Key from generating the same hash sequence if a user tries to reuse a base secret or uses the same one on different computers. Although Figure 10.3 shows only the hash stored in the password file entry, S/Key will also store the seed value and the hash sequence number. When an S/Key server issues a "challenge" containing the user's current sequence number, it will also display the seed used to generate the user's hashes.

S/Key users generally use software tokens to generate a one-time password. To use the software token, Cathy (the user in Figure 10.3) types in her base secret (her "password"), the sequence number, and the seed. The token applies the hash iteratively to generate the correct value in the sequence, and then displays the resulting hash. Cathy then copies the hash value into the waiting password prompt. Software tokens exist for Unix, Microsoft, and Macintosh. Where possible, the token software tries to detect an S/Key challenge so that it can automatically compute the correct one-time password. Where possible, the token software supports cut and paste to avoid typing errors when copying the challenge or the response. There is also a utility program that will print out hashes on paper for users that can't run a software token. Although it is technically possible to build S/Key hardware tokens, there don't appear to be any in commercial production. *see Note 4.*

As long as the server stores only the last hash in the sequence, and the user provides the next-to-last hash as a password, the attacker cannot easily retrieve a valid password. An attacker can't extract a hash from the password file and invert the computation in order to retrieve the previous hash in the sequence, or the base secret for that matter.

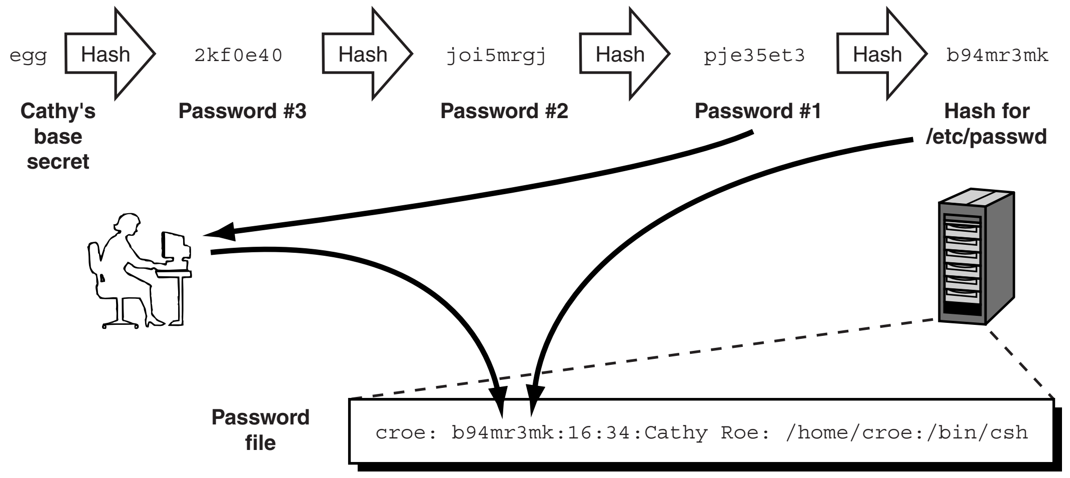

FIGURE 10.3: *Lamport's one-time password scheme, used by S/Key.* Cathy computes a sequence of passwords so that she can log on three times. She hashes her memorized base secret *four* times and stores the final hash in her password file entry. When she logs in next, she invokes the hash function *three* times to generate the right one-time password. The server verifies her password by hashing it (the fourth time) and matching the result with her password file entry. If they match, the latest password replaces the existing hash value in Cathy's password file entry, so that the process works the same when Password #2 is used.

10.2 Challenge Response Issues

This section examines two important concerns with respect to challenge response. First, there is the issue of user interaction, particularly for conventional challenge response tokens. Second, the traditional ANSI X9.9 standard is vulnerable to trial-and-error attack, since it is based on the DES algorithm.

USER INTERACTION

Challenge response works only with software and protocols that provide a way to transmit the challenge as part of the authentication process. Another problem is that it requires even more typing than other authentication methods. It is difficult to build software clients to simplify the typing problem.

Challenge response violates a very common assumption about authentication: that a server can always authenticate someone by collecting a user name and a password at the same time. This interferes with challenge response authentication because it provides no way to give the challenge. This is a problem with several important systems widely used today. For example, the Web page authentication extensions within the Hypertext Transfer Protocol (HTTP) bundle all of the authentication data in the same message that identifies the page being accessed. So there is no opportunity for the server to provide a random challenge to the person being authenticated.

If the software system is able to display the challenge to the user somehow, there is also the problem of getting that challenge into the challenge response computation. First, the person has to locate the challenge within some window on today's ever more cluttered display screens. Then the person has to type the challenge correctly. This is where the system can start to break down. The average person has enough trouble typing a password correctly; a hardware challenge response token may require the transcription of 15 to 20 digits, or more, including a PIN. This provides people with numerous opportunities to make a trivial typing mistake. Since password interfaces rarely echo the data typed, even when handling one-time passwords, the typing error won't be evident until the server rejects the attempt. Logging on becomes a very difficult process.

This is a situation in which software-based authentication clients provide an important benefit. Most users of Microsoft network authentication don't even know that challenge response authentication is taking place, since the underlying software handles it transparently.

Software versions of hardware tokens (the "soft" tokens described in Section 10.3) also provide usability benefits. For example, the workstation operator can often use window cut-and-paste operations to move the challenge and response between the token's window and the window being used to log on. This reduces the risk of typing errors, though less-sophisticated computer users are not as comfortable with this type of window manipulation.

KNOWN CIPHERTEXT ATTACK ON ANSI X9.9

One-time passwords generated with the X9.9 standard are vulnerable to attack by trial-and-error search. If an attacker collects several challenges and corresponding responses from a single X9.9 user, the attacker can try to deduce the user's base secret through a trial-and-error search of all possible DES keys. This is because X9.9 uses DES with the traditional 56-bit keys. As described in Section 5.3, DES has succumbed to several trial-and-error cracking demonstrations. The X9 committee has issued a report regarding this vulnerability.

A-63

see Note 5.

The only defense against the weakness in X9.9 is to migrate to a stronger mechanism. This is the recommendation of the X9 committee. Their analysis identifies nine existing standards that could be used to replace X9.9 functionality. Some alternatives are variants of existing techniques like DES encryption and one-way hashes but use longer secret keys. Another alternative would be to substitute AES for DES, to benefit from its higher performance and longer keys. Other alternatives rely on public key authentication techniques, like digital signatures, which are discussed in Chapter 13.

D-63

10.3 Password Token Deployment

This section examines three issues related to password token deployment. First, it looks at the use of "soft" tokens, that is, software implementations of the mechanisms that were traditionally implemented in hardware tokens. Next, it looks at the use of one-time passwords with multiple authentication servers. Finally, it briefly reviews the issue of using proprietary algorithms in security products.

SOFT TOKENS

Soft tokens are alternatives to hardware tokens that provide one-time passwords. Such tokens appear most often as software implementations of hardware tokens. In some cases, vendors also provide "codebooks" for one-time passwords. These approaches are discussed below.

Many existing hardware tokens, including both SafeWord and SecurID, are available as software tokens. The software version installs on the owner's remote computer, often a home computer or a laptop. Major vendors also provide palmtop versions of their software tokens to run on palmtop organizers like the Palm. The site generates the initial base secret, the same as for hardware tokens, but delivers them to individual users as data files. Once the owner installs the files correctly, the software token will work the same as a hardware token as far as logging on is concerned: the owner invokes the token to generate a one-time password, and then provides that password while logging on.

The principal benefit of software tokens is cost. While a hardware token system may cost as much as $100 per user to implement, including tokens, a software token system may cost only a third or a fourth as much.

Software tokens are much more vulnerable to theft than hardware tokens. Since the entire software token, including the base secret, is stored on a conventional personal computer, an attacker can easily make a copy of it. Unlike hardware tokens, the victim can still use the token even if a copy has been stolen. This is true of palmtop tokens, too, because desktop synchronization procedures like the Palm's "Hot Sync" automatically make back-up copies of the software token's files. Attempts to prevent copying of software token data files can cause serious reliability problems that often outweigh the intended security improvements.

PINs are an essential part of any software token implementation. They provide the principal, and often the only, protection against token theft. As described in Section 9.5, the software token should use the PIN as part of the base secret. The software token must not contain any information that can verify the PIN's value by itself. When an attacker tries to generate a one-time password with the software token, he should have no way of verifying the PIN or the password except by using the password to log on to the server.

Another type of soft token is the *codebook*, a paper listing of one-time passwords to use. Bob Bosen used this approach to implement the copy protection scheme that led to SafeWord. Codebooks may list passwords in their sequence of use, but more often they are used with a challenge response procedure. Typically, the authentication system will generate a separate codebook for each user that needs one. The codebook will be marked with the user name and have a range of validity dates to reduce the possible damage caused by one falling into the wrong hands.

A significant shortcoming of many soft tokens is that they make it possible for people to share their access permissions with other people by sharing the soft token's base secret. For example, if John has just been hired and he needs to use the home office system on his trip next week, someone else in the department (Cathy, for example) could share her soft token with him. By doing this, Cathy delegates her capabilities on the system to John. The system will associate all of John's actions with Cathy, even though John performed them. This might not be a serious problem in some environments, but the results can be expensive, as shown in the Citibank story that opened Section 9.3. Soft tokens are a good choice if the principal threat is password sniffing. They are not a good choice if the site needs to prevent delegation by individual users.

HANDLING MULTIPLE SERVERS

Today, most people don't complain about having to memorize a single password to use their computer: they complain about having to memorize a handful or two. If the computers in question use token-based one-time passwords, then password management needs extra planning. Tokens are expensive, and their proliferation is simply going to increase the number that get lost. The practical strategy is to allow people to use a single token to authenticate to any of their servers.

There are essentially three strategies. The first is to share the token's base secret among multiple sites. The second is to build slightly more complex tokens that can handle multiple base secrets. The third is to use indirect authentication through an authentication server. Consider the problem of Cathy, for example, who needs to talk to six different servers, each doing direct authentication.

While Cathy can do this simply by sharing her base secret, the results can be complicated and risky. In theory, she simply needs to share her base secret with the proprietors of those six different servers so that they can install the base secret in her password entry. If, however, she is using a synchronous token, then she could run into synchronization problems or open herself to a replay attack, or both. Separate servers will have to keep their own synchronization data for Cathy's token, and try to keep that information in sync. However, if Cathy ever has to resynchronize with one server (that is, send it a sequential pair of one-time passwords), then an attacker could sniff those and use them to log on to a different server. This is because the other server would treat the pair of passwords as a valid attempt to resynchronize the token.

Even if Cathy doesn't use a synchronous token, history has shown that secret sharing is a bad security strategy. It was a popular strategy in the encryption world for centuries but has fallen out of favor in the modern world of computer communications. If Cathy shares her secret with all those servers, then she faces several practical, security-oriented problems. First, there is the problem of transitive trust (see Attack A-40 in Section 6.5). Once she gives her base secret to an administrator, she has no way of ensuring that the administrator won't share it with someone else. This was probably the security flaw leading to the Citibank thefts discussed earlier.

An alternative to base secret sharing is to use tokens that store multiple base secrets. Such tokens are available from most major manufacturers, though they tend to be more expensive. To log on, Cathy tells her token which server she needs to use. Most tokens assign different numeric codes to different servers, and Cathy uses the appropriate code to choose the right base secret for a given server. Some tokens provide simple textual tags to choose from. In any case, it becomes the responsibility of individual server administrators to correctly program Cathy's token so that it includes a base secret known to their server. If the base secrets at Server A become compromised, the risks are somewhat contained. Cathy can still log on to other servers safely even if someone actually publishes Server A's base secrets, since Cathy's token uses different base secrets for the other servers. Cathy can add, delete, or otherwise update her base secrets for any of the servers without affecting her ability to authenticate to the others.

The third alternative is to use indirect authentication, so that the server Cathy contacts will in turn contact an authentication server to verify her one-time password. Again, most major manufacturers of one-time password tokens provide authentication servers. If all of Cathy's servers are part of a single site or organization, then an indirect authentication design provides the simplest solution. Chapter 11 talks about indirect authentication in more detail.

PROPRIETARY IMPLEMENTATIONS

Most commercial authentication token vendors sell products that contain unpublished, proprietary mechanisms to some degree. Many do not publish the low-level details of how their tokens perform their computations or how their servers handle resynchronization. Some vendors identify particular algorithms used (usually DES) but don't always explain exactly how DES is used or how keys are generated.

Today, many information security experts argue that the best products are those whose details are publicly available and have been subjected to scrutiny by the community at large. Some experts believe that customers take an unnecessary risk when they purchase a product whose details are proprietary. At the very least, public scrutiny by a diverse community of observers increases the likelihood that serious flaws will be caught, hopefully before any customers suffer damage. Public scrutiny increases the confidence customers can have in new security products. *see Note 6.*

Public scrutiny also has its drawbacks. There is the possibility that a flaw will be found and exploited before the vendor or other members of the community will find and fix it. In addition, we must recognize that secrecy does play a role in enhancing security: if attackers don't know the technical details of a system, then they have a much harder time attacking it.

Moreover, many vendors have thrived in the password token market without revealing various internal details of their products. There are two likely reasons: longevity and business model. The major token vendors have been in business since the 1980s and have established the reputation of their products. The existing base of satisfied customers provides new customers with the confidence they need to buy the products. In addition, many authentication vendors seek to make money on both hardware tokens and on server software. Both represent an installed base for existing customers that produces sales for the incumbent vendor. By keeping certain product details secret, the vendor prevents third parties from either selling compatible tokens to existing customers or selling replacements for the authentication server.

10.4 Evolving Windows Authentication

Microsoft's desktop authentication started with typical lock-screen techniques and, with the introduction of LAN Manager (often called LANMAN), embraced challenge response for authenticating network services. As the 1990s progressed, Microsoft developers took an evolutionary approach to authentication so that new products would be able to work with existing, installed software. Windows NT 4.0 brought a capability for indirect authentication to domain controllers (Chapter 11), which uses the acronym "NTLM" for "NT LAN Manager." Windows 2000 has incorporated the Kerberos protocol (Chapter 12). Although the LANMAN and NTLM mechanisms are being replaced by Kerberos, it is still essential to understand the older protocols since they are often used with older, installed software.

see Note 7.

For the most part, Microsoft's products use network-based authentication as a way to control access to network resources. Users on workstations manipulate the data on their workstations, but the final result is often printed on a network printer or stored on a network server. Unlike conventional passwords, or even token-based challenge response, Microsoft's network authentication involves a relatively complicated protocol that is embedded in the desktop client software. The protocol finds its way into file and printer services by transmitting authentication data in Server Message Block (SMB) protocol messages that provide such services across the local network.

Microsoft's LAN and NT password technology incorporated two important features. First, all password databases consisted of hashed passwords. Second, all authentication used challenge response techniques to thwart password sniffing. These features, combined with the evolution from LANMAN to NTLM, yielded a relatively complicated password environment. We look at Windows challenge response in Sections 10.5 and 10.6.

Windows systems store password hashes in a system file and, where possible, provide some degree of protection against attackers stealing the password file. Modern Windows systems include a special storage area called the *Registry*. The Security Accounts Manager (SAM) database in the Registry maintains entries for all authorized users, and stores their hashed passwords as well. Windows NT enforces user-based access restrictions on Registry file entries, and the SAM entry is heavily restricted. While this does not prevent all attempts to extract the Windows password file, it increases the difficulty of such an attack.

Windows authentication has proven to be vulnerable to two types of attacks: off-line password crackers that attack the SAM database, and crackers that work on intercepted challenge response pairs. Basic SAM attacks evolved as a result of work by the Samba Project, which produced a public domain package for sharing files between Unix servers and NT clients. The Unix server needed to extract copies of users' NT password hashes in order to synchronize passwords. Attackers also used the tool, and variants of it like *pwdump*, to extract hashes to perform trial-and-error guessing attacks on them. ***see Note 8.***

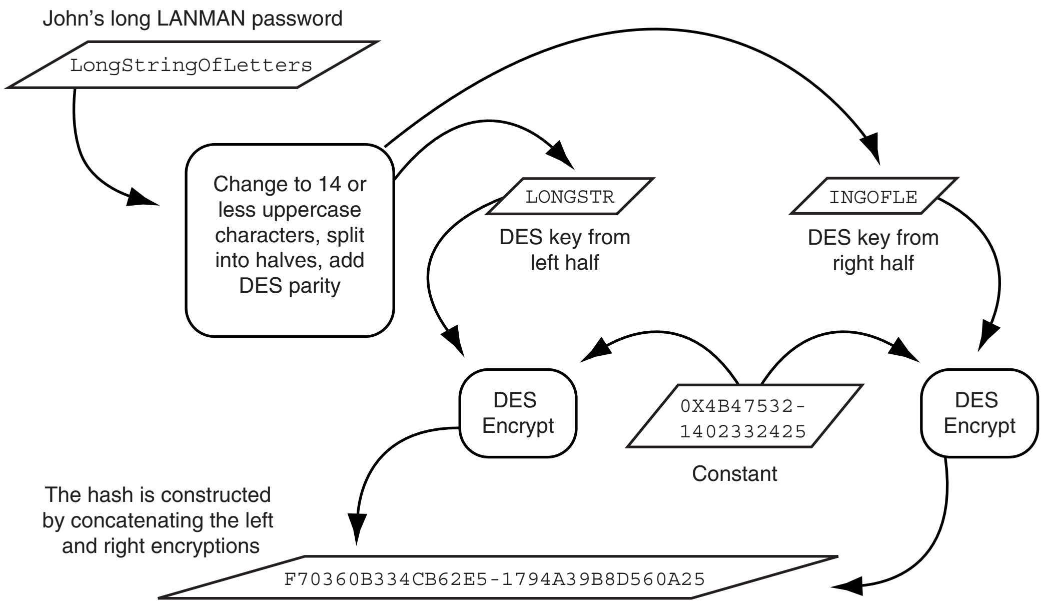
John's long LANMAN password
LongStringOfLetters
Change to 14 or less uppercase characters, split into halves, add DES parity
LONGSTR
DES key from left half
INGOFLE
DES key from right half
DES Encrypt
0X4B47532-1402332425
Constant
DES Encrypt
The hash is constructed by concatenating the left and right encryptions
F70360B334CB62E5-1794A39B8D560A25

FIGURE 10.4: *Generating a LANMAN hash.* John's typed password is changed to be 14 characters long and any lowercase characters are changed to uppercase. The result is broken into two 56-bit chunks, and each is used to encrypt a constant. The two 64-bit results are combined to produce the hash.

LANMAN HASHING

Figure 10.4 illustrates how the LANMAN hash function transforms a lengthy password belonging to John Doe. First, the function changes the password into a 14-character string, adding or removing characters as needed. Next, it converts all the characters to uppercase. This is a good move from a usability standpoint, since it allows John to make mistakes with the shift key and the system will still recognize his password. However, it also reduces the password's entropy.

Then the function splits the result into two seven-byte chunks, and uses each chunk as a 56-bit key for DES encryption. Each key separately encrypts a 64-bit constant. Unix uses DES in essentially the same way (Figure 2.5 in Section 2.3), but LANMAN omits the "salt." The encrypted results are concatenated into a single string to produce the final hash result. ***see Note 9.***

When creating or changing John's password, the system computes the hash and saves it in the SAM database. When checking a password used for a local logon, the system hashes the password John types in and compares the hashed result with the hash value in the SAM database.

ATTACKING THE LANMAN HASH

Although the LANMAN hash may look complicated to the untrained eye, it carries two critical weaknesses. First, the procedure simplifies a trial-and-error password search by encrypting the password in two completely independent seven-character chunks. Second, the procedure limits characters to a single case, which reduces the password's entropy. Both of these problems reduce the average attack space of a hashed LANMAN password and make it practical to attack. *see Note 10.*

To appreciate the first problem, let's look at the average attack space we should achieve if we have 14-character alphabetic passwords. Using the shortcut computation we saw in Section 2.7, we compute the password space like this:

$26^{14} \approx 10^{19}$,

which yields an average attack space of 65 bits.

Unfortunately, LANMAN's arrangement of seven-character chunks allows attackers to search for the password piecemeal instead of having to try all permutations of every character. Figure 10.4 illustrates the problem. Note how the procedure splits the password into the separate pieces "LONGSTR" and "INGOFLE" and encrypts them separately. The attacker can crack the password hash by looking for each seven-byte chunk separately. In other words, the attacker recovers a 14-letter password by testing this many different possibilities: ***A-64***

$26^7 + 26^7$,

or $2 \times 26^7 \approx 10^{10}$,

which yields an average attack space of only 32 bits.

The problem of password chunks is not new. A celebrated example occurred on the TENEX timesharing system in the 1970s. The

TENEX password checking program maintained a table of plaintext passwords and would check the passwords a character at a time. As soon as Tenex encountered a character in the password that didn't match, it returned a failure without checking the rest of the characters. A clever person realized that the Tenex memory management system would send a signal to his program if the password checking program stepped past the end of his program's available memory. He then used this fact to construct a password cracking program that treated each character as a chunk, and induced Tenex to tell his program whether a particular chunk matched or not. ***see Note 11.***

We solve this problem by making sure we handle the entire password as a single chunk, instead of handling it in two or more pieces. In Tenex, the system programmers solved the problem by having the password checking program copy the password into system memory before checking it. Windows NT solved the LANMAN problem by using a one-way hash algorithm to combine the entire 14-character password in a single cryptographic operation, as is described in Section 10.6.

The second LANMAN problem is that attackers don't need to try all the permutations of upper- and lowercase letters in a password; they can limit the search to uppercase letters and still match the password. Users can choose passwords with a mixture of upper- and lowercase letters, but the LANMAN hashing procedure eliminates the difference when generating the hash value. If attackers had to search both upper- and lowercase characters to crack a LANMAN hash, then the average attack space increases from 32 bits to 39 bits, which increases the attackers' effort by over a hundredfold.

When we combine the shortcomings of LANMAN passwords, we find a straightforward procedure for cracking them. We start by attacking the second half of the hash value. In many sites, most people's passwords will be shorter than 14 characters (most will be *a lot* shorter). If the password is seven characters or less, then the second half of the hash will match the hash value for all nulls. If the password is eight to ten characters long, the second hash will contain only one to three characters, and pose a very simple cracking job. In some cases, the final characters suffixing the password may yield a hint as to the password's first seven characters. For example, the letters "abra" might be the suffix of the password "abracadabra."

A more sophisticated attack might search a dictionary for words matching the suffix and then try just those words.

The second half of the hash is of course the easy part. But the first half will probably fall to some other well-known attack. At worst, a dedicated computer could work through all of the possible passwords and eventually crack it. Although this might not be practical for a single personal computer, a motivated attacker can apply several computers to the task to solve the problem more rapidly.

The problem with LANMAN passwords is tied to the construction of the hash procedure. A better design would provide significantly better security while incurring the same amount of processing overhead. The good news, however, is that at least LANMAN provides some measure of protection to its passwords. For many sites, poor protection is better than none at all, as long as they keep in mind the risks they face.

PLAINTEXT PASSWORDS ON WINDOWS

LANMAN did not always use challenge response passwords, and other LAN products persist that use plaintext passwords. This is particularly important when interoperating with other products. For example, some versions of the Samba package for hosting Windows-compatible network services on Unix used plaintext passwords. Windows products, including NT, provide the necessary hooks to allow a certain degree of automation for logging on to services that require plaintext passwords. Administrators may set a special configuration flag in the Windows Registry to allow the use of plaintext passwords. NT makes it difficult to use plaintext passwords so that administrators are alert to the potential risk of password sniffing.

The support of plaintext passwords does cause a potential problem called the *downgrade attack*. In this attack, the attacker tricks a client into believing that it should provide plaintext authentication to a server. The client responds by automatically providing the password in plaintext instead of hashing the password and using the challenge response protocol. ***A-65***

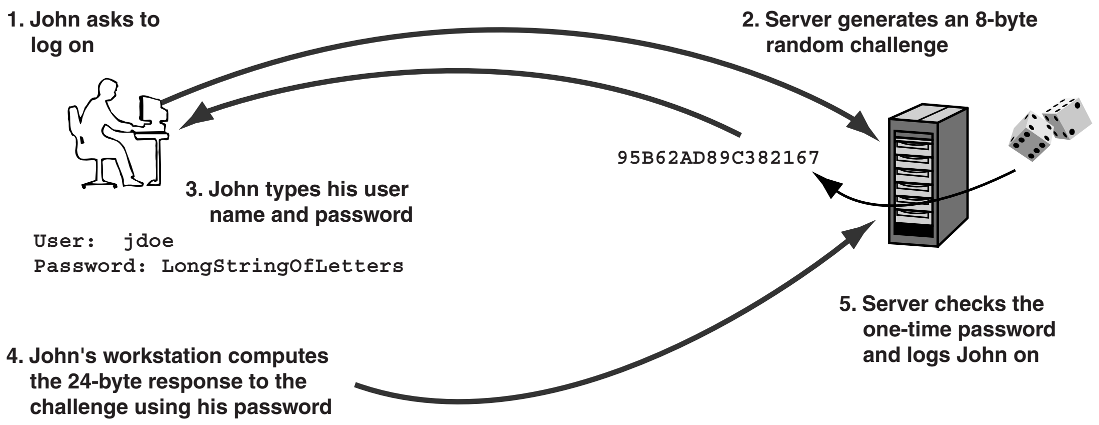
1. John asks to log on
2. Server generates an 8-byte random challenge
95B62AD89C382167
3. John types his user name and password
User: jdoe
Password: LongStringOfLetters
4. John's workstation computes the 24-byte response to the challenge using his password
5. Server checks the one-time password and logs John on

FIGURE 10.5: *Windows challenge response authentication.* The challenge response protocol is built directly into the logon software for Windows LANMAN and NT. To log on, John simply types in his user name and secret password. The workstation automatically intercepts the challenge from the server, computes the response, and sends it back without user intervention.

10.5 WINDOWS CHALLENGE RESPONSE

Unlike token-based challenge response, the Windows logon protocol automatically intercepts the challenge and generates the response automatically based on the owner's password. Figure 10.5 illustrates the procedure. When John asks to log on to a server, the server replies with a randomly generated eight-byte nonce. If it hasn't done so already, John's workstation then prompts John for his user name and password. Upon receiving the password, the workstation hashes it so that it does not need to store the plaintext password. The workstation will generally keep a copy of the password hash so that John won't have to type it in again before he logs

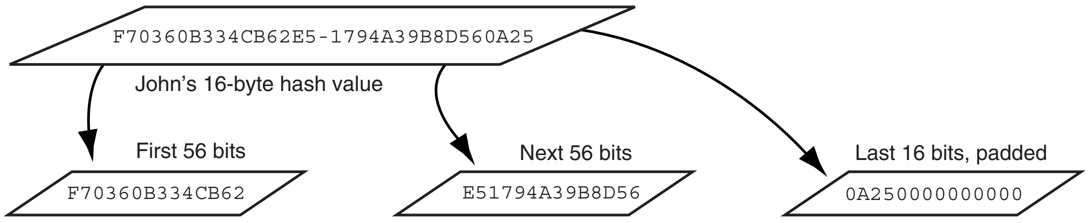

FIGURE 10.6: *Keys for a Windows response to a challenge.* The user's hash value is broken into three 56-bit keys, padded with nulls. These are used individually to encrypt the challenge

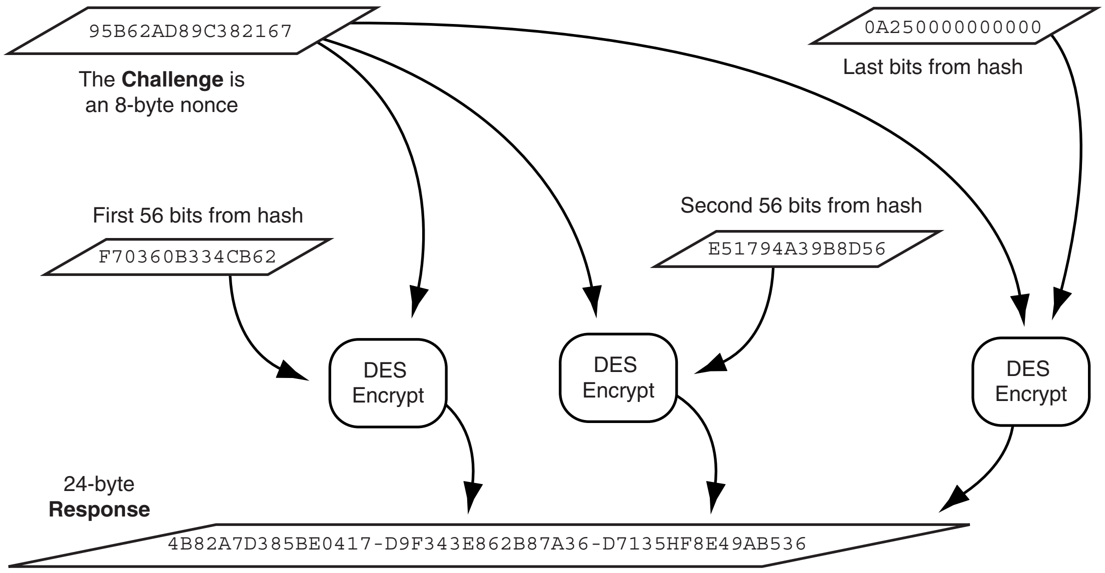

FIGURE 10.7: *Generating a Windows response to a challenge.* Parts from the user's hash are used as DES keys, and the nonce is encrypted once with each of the keys. The response consists of the nonce encrypted with each key.

off. Finally, the workstation computes the response by applying DES encryption to the nonce three times. *see Note 12.*

Figures 10.6 and 10.7 illustrate how Windows software computes the response for an authentication challenge. LANMAN introduced this procedure and NTLM uses a variant of it. First, the procedure takes the user's 128-bit hash value and produces three 56-bit pieces (Figure 10.6). Then the procedure encrypts the nonce three times, using each piece of the hash as a DES key. The procedure combines results of the three encryptions into the 24-byte response.

ATTACKING WINDOWS CHALLENGE RESPONSE

There are essentially two approaches to attacking the Windows challenge response procedure. First, an attacker can search for a password that matches a challenge response pair. Second, the attacker can use a copy of someone's hashed password to masquerade as that person. Both attacks are based on theoretical properties of the challenge response protocol and, as of this writing, are not in any set of tools that attackers are known to use.

The first attack combines the classic sniffer attack with an off-line dictionary attack. Actually, the sniffer attack is slightly more complicated because the attacker must intercept both the server's challenge and the client's response. If the attacker intercepts several challenges and responses, each must be correctly paired up, and each pair must be associated with the user that generated the response. The attacker takes a set of challenge response pairs associated with a particular user and performs a password search attack. For example, the attacker could use a dictionary to generate candidate passwords, generate the hash for each password, compute the appropriate response, and match it against the intercepted response.

The second attack simply exploits a password hash that was stolen from a SAM database. As shown in Figure 10.7, the challenge response relies entirely on the password hash value and not on the password itself. In other words, the hash value serves as a *password equivalent*. The actual base secret used for authentication is the password hash value, not the password.

If an attacker has a copy of some user's password hash, the attacker can correctly respond to a server's challenge and masquerade as that other user. This requires a subverted Windows client **A-66** which allows the attacker to insert the appropriate user name and password hash into the password database, and then log on without actually supplying a password. This is not normal behavior for Windows clients. However, traditional Windows clients (non-NT systems prior to Windows 2000) have no real protection against attackers patching their software to behave in odd and inappropriate ways.

10.6 Windows NTLM Authentication

Microsoft Windows NT 4.0, the last system labeled "NT," supports three separate types of authentication, which are termed "local," "domain," and "remote." Local authentication has the same meaning as the local authentication pattern discussed in Chapter 4: a person logs on to a device directly and does not establish a remote connection. Domain authentication corresponds to the direct authentication pattern and represents the case in which a person uses his or her personal computer to log on to a different computer across a network. LANMAN authentication is also an example of this. Remote authentication corresponds to the indirect authentication pattern, in which a client logs on to a server who in turn relies on a different server (the NT *domain controller*) to verify a user's challenge response. When we refer to "NTLM" authentication, we generally are referring to the latter two types, which represent NT network authentication.

Developers of Windows NT saw an opportunity to make major improvements over LANMAN authentication. NT supported an extended character set which would greatly increase the number of possible passwords. Several new cryptographic algorithms were available that could reduce computational overhead while maintaining or even improving the level of security provided.

This led Windows NT to use a new password hashing procedure, illustrated in Figure 10.8. NT retains the 14-character limit on password size, but allows people to use any character in the extended Unicode character set. Passwords are read in and stored as a sequence of fourteen 16-bit Unicode characters. To create the 128-bit password hash, NT uses Message Digest #4 (MD4), a commercial hash algorithm developed by Ron Rivest (a newer algorithm, MD5, is widely used in Internet protocols). This improved hash is usually called the *NTLM hash*.

D-64

see Note 13.

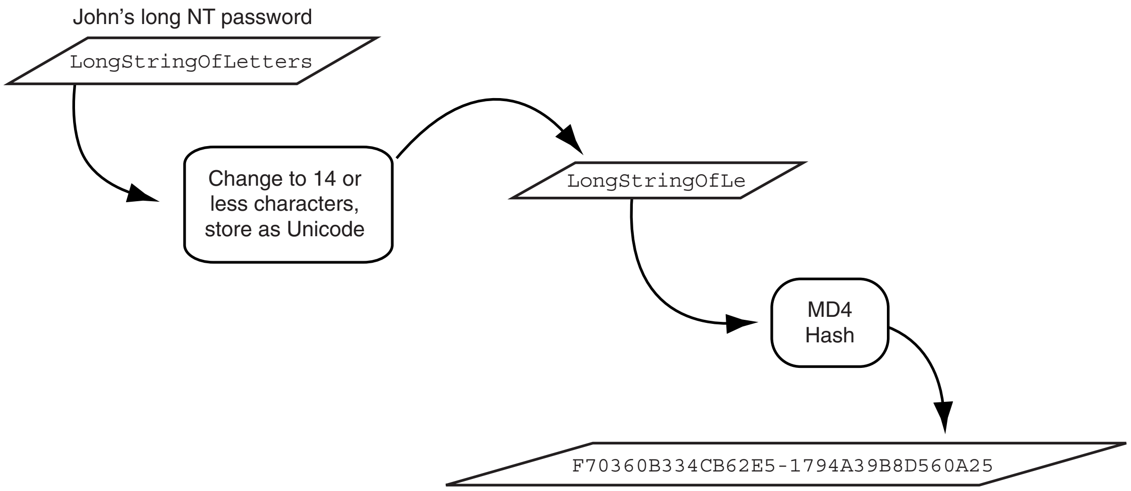
John's long NT password
LongStringOfLetters
Change to 14 or less characters, store as Unicode
LongStringOfLe
MD4 Hash
F70360B334CB62E5-1794A39B8D560A25

FIGURE 10.8: *Generating the NTLM hash.* John's typed password is changed to be 14 characters long and each character is converted to its 16-bit Unicode representation. MD4 hashes this string of characters into a 128-bit hash value that is used as the NT password hash.

Although NTLM authentication uses an improved hash function for encoding passwords, it still uses the challenge response protocol shown in Figure 10.6. However, NTLM authentication acquired some odd complications in order to retain compatibility with LANMAN. NTLM authentication requires *both* an NTLM hash and a LANMAN hash. Each user password entry in the SAM database contains two password hashes: one computed using the NTLM hash procedure and a second one computed using the LANMAN procedure. When an NT client performs a challenge response authentication, it computes two responses: one using the NTLM hash and another using the LANMAN hash. This approach allows NT systems to interoperate with much older LAN software, but it also opens up some unfortunate weaknesses in NTLM authentication. For many sites, this interoperability feature manages to negate the security improvements brought about by the redesigned NTLM hash procedure.

ATTACKING THE NT PASSWORD DATABASE

Attackers can recover passwords by attacking the hashes stored in NT's SAM database. The first step in this attack is to retrieve hash values from the database. The second step is to crack the hash values once they have been intercepted. We examine the second step first.

Each user entry in NT's password database contains two separate hash values: one computed using the newer NTLM hash procedure and another computed using the older LANMAN procedure. Both hashes are computed from the user's secret password.

The attack takes place in two steps. The first step is to attack the LANMAN password as described in Section 10.4. This recovers the **A-67** password but does not indicate which letters are uppercase and which are lowercase, since the LANMAN hash maps everything to uppercase. The attacker must recover the case information; otherwise, the attacker can't use the password to log directly into an NT system.

The second step is to recover the NT password. To do this, the attacker must find which letters in the password are supposed to be in uppercase and which in lowercase. The attacker generates every possible permutation of case changes for the letters in the password, computes the NTLM hash for each permutation, and compares the result against the hash value from the SAM database. The two values match when the attacker finds the right choices of upper- and lowercase letters. *see Note 14.*

Unlike less-sophisticated Windows products, NT provides user-oriented access control and can use it to block access to the SAM database (Defense D-20 in Chapter 2). This is the essential defense against attacks on NT passwords. Unfortunately, this is not foolproof. Interoperability requirements often lead sites to export password hashes and install them on other systems. Any time a sensitive file is copied, the risk of its disclosure increases. Furthermore, history has shown that user-based access control isn't enough to ensure the secrecy of a file on a multiuser system.

To reduce attacks on the SAM database, Microsoft released a patch to Windows NT to provide encryption of the SAM database. *D-65* This mechanism, called the "System Key" or SYSKEY fix, uses a secret key to encrypt the SAM database. The computer's administrator must provide the key to the system somehow, and then the system decrypts the database and allows users to log on. *see Note 15.*

The system key itself poses another security quandary: where should it stay when the computer is shut down? The key should not be accessible by attackers, nor should copies sit around the computer room in case administrators are careless or have been subverted. A hard-to-crack encryption key should be just about impossible for someone to memorize. On the other hand, the key should be available to the computer at all times so that it may reboot itself without operator intervention. Microsoft provided three alternatives: type in the key at boot time, "hide" the key in the Registry, or store the key on a removable diskette. Each of these techniques has its risks and benefits.

Typing the key in at boot time is probably the least appealing because it forces an administrator to be present whenever the computer reboots. Furthermore, the key will have to be provided to every administrator that might be called upon to reboot the computer, and this increases the risk of some types of misuse or disclosure. As a practical matter, many administrators might keep the key written down near the keyboard that is used to boot the computer.

If the key is hidden in the Registry, then the computer can reboot itself without operator intervention. This is very important for availability. On the other hand, it is difficult or perhaps even impossible to hide this type of key in a computer system. There are always ways of locating such keys. Once attackers figure out where the key is located, they can try to extract it from a system backup. A sufficiently privileged user could probably extract the key directly from the Registry.

Storing the key on a diskette has the same advantage as hiding the key in the registry: it becomes possible to reboot the computer without operator intervention. Furthermore, the key becomes somewhat easier to control if it resides on a diskette. Since it is not on the system volume, it will not be copied onto back-up tapes, and this reduces the risk of attackers extracting the key from a backup. Since the diskette is portable, the administrators can remove it and lock it up if necessary. On the other hand, a careless or subverted administrator could make a copy of the diskette and allow it to fall into the wrong hands. Also, a sufficiently privileged administrator could probably extract the key off of the diskette as long as the diskette is installed in the computer's drive. ***see Note 16.***

Many NT systems provide an extra opportunity for attackers to steal the SAM database: the recovery disk. NT systems allow administrators to maintain crucial system configuration files on a diskette called the *recovery disk*. One of the crucial files is, of course, the SAM database. Moreover, many systems maintain a copy of the recovery disk's files on-line to simplify the process of keeping the disk up to date. Attackers have been able to recover the SAM database by stealing it from the recovery disk area of a domain controller. ***A-68*** There are two strategies for preventing this attack: (1) put access protection on the recovery disk files, or (2) do not store the files on-line except when actually constructing a recovery disk.

ATTACKING NTLM CHALLENGE RESPONSE

The NTLM challenge response may be attacked in essentially the same way as the LANMAN challenge response. Attackers can try to recover passwords by intercepting matching pairs of challenges and responses and cracking the password. Attackers can also use subverted clients to exploit stolen password hashes. However, the principal difference for NT is in the strategy used to crack passwords from intercepted challenge response pairs.

Since the NTLM protocol transmits both the NTLM response and the LANMAN response, attackers can mount an attack similar to the one against the NT password database, but target it against challenge response pairs. Again, the attack takes place in two steps. The first step attacks the LANMAN challenge response as described in Section 10.5. The next step uses the candidate LANMAN password as the starting point and searches for the NT password by trying permutations of upper- and lowercase characters.

The principal defense is to disable the weaker mechanism so that attackers can't exploit its weaknesses. In this case, NT must be configured to disable LANMAN authentication. This eliminates the LANMAN hashes from the challenge response transactions. NTLM challenge response pairs may be vulnerable to dictionary attacks, but the stronger hash makes them much harder to attack. Originally, there was no way to eliminate the LANMAN weakness, but Microsoft produced a patch to optionally disable LANMAN authentication to increase security. ***D-66***

10.7 Summary Tables

TABLE 10.2: *Attack Summary*

Attack	**Security Problem**	**Prevalence**	**Attack Description**
A-63. Trial-and-error attack on X9.9	Masquerade as someone else	Physical and Sophisticated	Intercept several one-time passwords from an X9.9 user, crack the base secret using a DES cracker
A-64. Crack passwords in sections	Recover a user's password	Common	Cracks the password in independent parts, so attack is linear by parts instead of geometric
A-65. Force use of plaintext password	Recover a user's password	Common	Forces server to ask the user for a plaintext password so that it can be sniffed on the network
A-66. Logged in hash substitution	Masquerade as someone else	Sophisticated	Embed a stolen hash in the SAM database and subvert NT so that the user appears logged in

TABLE 10.2: *Attack Summary (Continued)*

Attack	Security Problem	Prevalence	Attack Description
A-67. Use LAN-MAN hash to crack NT hash	Recover a user's password	Common	Use password cracking software that exploits weaknesses of LAN-MAN hash to crack NT hashes
A-68. Copy hashes from NT recovery files	Recover a user's password	Trivial	Extract password hashes from the NT recovery disk files and use a cracking program on them

TABLE 10.3: *Defense Summary*

Defense	Foils Attacks	Description
D-63. Use longer cryptographic keys	A-63. Trial-and-error attack on X9.9	Replace existing technical measures with mechanisms that use longer cryptographic keys so that they better resist trial-and-error attacks
D-64. Interdependent hash computation	A-64. Crack passwords in sections	Every part of the password hash value depends on the value of every part of the password. There is no way to crack part of the password
D-65. Database encryption	A-67. Use LANMAN hash to crack NT hash A-68. Copy hashes from NT recovery files	Encrypt the entire password database so that attackers cannot attack the hashes
D-66. Disable weaker authentication	A-65. Force use of plaintext password	Configure the system to forbid the use of weaker mechanisms provided for backward compatibility

RESIDUAL ATTACK

A-66. Logged in hash substitution—the problem is that the Windows hash *is* the base secret. We foil this attack if we eliminate the need to store the base secret within the vulnerable Windows operating system. Kerberos provides one approach, which is to use a temporary base secret. We examine Kerberos in Chapter 12.