Limitations of the Kerberos Protocol

*S.M. Bellovin**
smb@ulysses.att.com
M. Merritt†
mischu@allegra.att.com

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

The Kerberos authentication system, a part of MIT's Project Athena, has been adopted by other organizations. Despite Kerberos's many strengths, it has a number of limitations and some weaknesses. Some are due to specifics of the MIT environment; others represent failures in the protocol design. We discuss a number of such problems, and present solutions to some of them. We also demonstrate how special-purpose cryptographic hardware may be needed in some cases.

1. INTRODUCTION

The Kerberos authentication system[Stei88] was introduced by MIT to meet the needs of Project Athena. It has since been adopted by a number of other organizations for their own purposes, and is being discussed as a possible standard. In our view, these decisions may be premature. Kerberos has a number of limitations and weaknesses; a decision to adopt or reject it cannot properly be made without considering these issues. Some improvements can be made within the current design. Support for optional mechanisms would extend Kerberos's applicability to environments radically different from MIT.

The limitations fall into several categories. Some stem from the Project Athena environment. Kerberos was designed for that environment; if the basic assumptions differ, the authentication system may need to be changed as well. Other limitations are simply deficiencies in the protocol design. Some of these are corrected in the proposed Version 5 of Kerberos,[Kohl89] but not all. Even the solved problems merit discussion, since the code for Version 4 has been widely disseminated. Finally, some problems with Kerberos are not solvable without employing special-purpose hardware, no matter what the design of the protocol. We will consider each of these areas in turn.

We wish to stress that we are not suggesting that Kerberos is useless. Quite the contrary — an attacker capable of carrying out any of the attacks listed here could penetrate a typical network of UNIX® systems far more easily. Adding Kerberos to a network will, under virtually all circumstances, significantly increase its security; our criticisms focus on the extent to which security is improved. Further, we recommend changes to the protocols that substantially increase security.

Beyond its specific utility in production, Kerberos serves a major function by focusing interest on practical solutions to the network authentication problem. The elegant protocol design and wide availability of the code has galvanized a wide audience. Far from a condemnation, this critique is intended to contribute to an understanding of Kerberos' properties and to influence its evolution into a tool of greater power and utility.

---

* Author's address: Steven M. Bellovin, Room 3C-536B, AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974.

† Author's address: Michael Merritt, Room 3D-458, AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974.

Several of the problems we point out are mentioned in the original Kerberos paper or elsewhere.[Davi89] For some of these, we present protocol improvements that solve, or at least ameliorate, the problem; for others, we place them squarely in the context of the intended Kerberos environment.

2. WHAT'S A KERBEROS?

Before discussing specific problem areas, it is helpful to review the Kerberos Version 4 protocols. Kerberos is an *authentication* system; it provides evidence of a *principal*'s identity. A principal is generally either a user or a particular service on some machine. A principal consists of the three-tuple

$$< primaryname,\ instance,\ realm >.$$

If the principal is a user — a genuine person — the *primary name* is the login identifier, and the *instance* is either null or represents particular attributes of the user, i.e., `root`. For a service, the machine name is used as the primary name and the service name is used as the instance, i.e., `myhost.rlogin`. The *realm* is used to distinguish among different authentication domains; thus, there need not be one giant — and universally trusted — Kerberos database serving an entire company.

Kerberos principals may obtain *tickets* for services from a special server known as the *ticket-granting server*, or *TGS*. A ticket contains assorted information identifying the principal, encrypted in the private key of the service. (Notation is summarized in Table 1.)

$$T_{c,s} = \{s,\ c,\ addr,\ timestamp,\ lifetime,\ K_{c,s}\}K_s$$

Since only Kerberos and the service share the private key K_s, the ticket is known to be authentic. The ticket contains a new private session key, $K_{c,s}$ known to the client as well; this key may be used to encrypt transactions during the session.[1]

To guard against *replay attacks*, all tickets presented are accompanied by an *authenticator*:

$$A_c = \{c,\ addr,\ timestamp\}K_{c,s}$$

This is a brief string encrypted in the session key and containing a timestamp; if the time does not match the current time within the (predetermined) clock skew limits, the request is assumed to be fraudulent.

For services where the client needs bidirectional authentication, the server can reply with

$$\{timestamp + 1\}K_{c,s}$$

This demonstrates that the server was able to read *timestamp* from the authenticator, and hence that it knew $K_{c,s}$; that in turn is only available in the ticket, which is encrypted in the server's private key.

Tickets are obtained from the TGS by sending a *request*

$$s,\ \{T_{c,tgs}\}K_{tgs},\ \{A_c\}K_{c,tgs}$$

In other words, an ordinary ticket/authenticator pair is used; the ticket is known as the *ticket-granting ticket*. The TGS responds with a ticket for server s and a copy of $K_{c,s}$, all encrypted with a private key shared by the TGS and the principal:

$$\{\{T_{c,s}\}K_s, K_{c,s}\}K_{c,tgs}$$

That key, $K_{c,s}$, and the ticket-granting ticket itself, are obtained from Kerberos at session-start time. The client sends a message to Kerberos with a principal name; Kerberos responds with

$$\{K_{c,tgs},\ \{T_{c,tgs}\}K_{tgs}\}K_c$$

The client key K_c is derived from a non-invertible transform of the user's typed password. Thus, all

1. Technically speaking, $K_{c,s}$ is a *multi-session key*, since it is used for all contacts with that server during the life of the ticket.

TABLE 1. Notation

c	client principal
s	server principal
tgs	ticket-granting server
K_x	private key of ‘‘x’’
$K_{c,s}$	session key for ‘‘c’’ and ‘‘s’’
$\{info\}K_x$	‘‘$info$’’ encrypted in key K_x
$T_{c,s}$	Ticket for ‘‘c’’ to use ‘‘s’’
A_c	Authenticator for ‘‘c’’
tgs	ticket-granting service
$addr$	client’s IP address

privileges depend ultimately on this one key.

Note that servers must possess private keys of their own, in order to decrypt tickets. These keys are stored in a secure location on the server’s machine.

3. THE KERBEROS ENVIRONMENT

The Project Athena computing environment consists of a large number of more or less anonymous workstations, and a smaller number of large server machines. The servers provide volatile file storage, print spooling, mailboxes, and perhaps some computing power; the workstations are used for most interaction and computing. Generally, they possess local disks, but these disks are effectively read-only; they contain no user data, and are replaced as needed. Furthermore, they are not physically secure; someone so inclined could remove, read, or alter any portion of the disk without hindrance.

A consequence of this is that the primary need is for user-to-server authentication. That is, when a user sits down at a workstation, that person needs access to private files residing on a server. The workstation itself has no such files, and hence has no need to contact the server or even to identify itself.

This is in marked contrast to a typical UNIX system’s view of the world. Such systems do have an identity, and they do own files. Assorted network daemons transfer files in the background, clock daemons perform management functions, electronic mail and news arrives, etc. If such a machine relied on servers to store its files, it would have to assert, and possibly prove, an identity when talking to these servers. The Project Athena workstations are neither capable nor in need of such; they in effect function as very smart terminals with substantial local computing power, rather than as full computer systems.[2]

What does this mean for Kerberos? Simply this: Kerberos is designed to authenticate the end-user — the human being sitting at the keyboard — to some number of servers. It is not a peer-to-peer system; it is not intended to be used by one computer’s daemons when contacting another computer. Attempting to use Kerberos in such a mode can cause trouble.

We make this statement for several reasons. First and foremost, typical computer systems do not have a secure key storage area. For one thing, a plaintext key must be used in the initial dialog with the key distribution center. But storing plaintext keys in a machine is generally felt to be a bad idea;[Morr79] if a Kerberos machine key is compromised, the intruder can likely impersonate any user on that computer. Additionally, the session keys returned by the TGS cannot be stored securely; of necessity, they are stored in some area accessible to `root`. Thus, if the intruder can crack the protection mechanism on the local computer — or, perhaps more to the point, work around it for some limited purposes — all current session keys can be stolen. This is less serious than a breach of the primary Kerberos key, of course, since session keys are limited in lifetime and scope; nevertheless, one does not wish these keys

2. We regard this as a feature, not a bug.

exposed.

This points out a second flaw when multi-user computers employ Kerberos, either on their own behalf or for their users: the cached keys are accessible to attackers logged in at the same time. In a workstation environment, only the current user has access to system resources; there is little or no need even to enable remote login to that workstation. There are many reasons for this; a consequence, though, is that the intruder simply cannot approach the safe door to try to pick its lock.[3] Only when the legitimate user leaves can the attacker attempt to find the keys. Rather, the attacker must sift through the debris; Kerberos attempts to wipe out old keys at logoff time. With a multi-user computer, on the other hand, an attacker has concurrent access to the keys if there are flaws in the host's security.

There are two other minor flaws in Kerberos directly attributable to the environment. First, there is some question about where keys should be cached. Since all of the Project Athena machines have local disks, the original code used `/tmp`. But this is highly insecure on diskless workstations, where `/tmp` exists on a file server; accordingly, a modification was made to store keys in shared memory. However, there is no guarantee that shared memory is not paged; if this entails network traffic, an intruder can capture these keys.

Finally, the Kerberos protocol binds tickets to IP addresses. Such usage is problematic on on multi-homed hosts (i.e., hosts with more than one IP address). Since workstations rarely have multiple addresses, this feature — intended to enhance security — was not a problem at MIT. Multi-user hosts often do have multiple addresses, however, and cannot live with this limitation. This problem has been fixed in Version 5.

4. PROTOCOL PROBLEMS

4.1 Caching Authenticators

The Kerberos protocol is not as resistant to penetration as it should be. A number of weaknesses are apparent; the most serious is its use of an authenticator to prevent replay attacks.

The authenticator relies on use of a timestamp to guard against reuse. This is problematic for several reasons. The claim is made that no replays are likely within the lifetime of the authenticator (typically five minutes). This is reinforced by the presence of the IP address in both the ticket and the authenticator. We are not persuaded by this logic. An intruder would not start by capturing a ticket and authenticator, and then develop the software to use them; rather, everything would be in place before the ticket-capture was attempted. Attacks involving multiple machines and jammed Ethernet[4] cables would be quite possible.

Further, the proposed Version 5 of Kerberos anticipates alternative communication protocols in which such replays may be trivial to implement. If Kerberos is to be considered as a general-purpose utility, it must make few security-critical assumptions about the underlying network, and those must be explicit.

It has been suggested that the proper defense is for the server to store all live authenticators; thus, an attempt to reuse one can be detected. For several reasons, we do not think that this solves the problem. First, on UNIX systems it is difficult for TCP-based[Post81] servers to store authenticators. Servers generally operate by forking a separate process to handle each incoming request. The child processes do not share any memory with the parent process, and thus have no convenient way to inform it — and hence any other child servers — of the value of the authenticator used. There are a number of obvious solutions — pipes, authenticator servers, shared memory segments and the like — but all are awkward, and some even raise authentication questions of their own. To date, we know of no multi-threaded server implementation which caches authenticators.

3. On Project Athena machines, remote access to most workstations is disabled by the *toehold* service.

4. Ethernet is a registered trademark of Xerox Corporation.

UDP-based[Post80] query servers can store the authenticators more easily; however, they might have problems with legitimate retransmissions of the client's request if the answer was lost. (UDP does not provide guaranteed delivery; thus, all retransmissions happen from application level, and are visible to the application.) Legitimate requests could be rejected, and a security alarm raised inappropriately. One possible solution would be for the application to generate a new authenticator when retransmitting a request; were it not for the other weaknesses of the authenticator scheme, this would be acceptable.

4.2 Secure Time Services

As noted, authenticators rely on machines' clocks being roughly synchronized. If a host can be misled about the correct time, a stale authenticator can be replayed without any trouble at all. Since some time synchronization protocols are unauthenticated,[Post83, Mill88] and hosts are still using these protocols despite the existence of better ones,[Mill89] such attacks are not difficult.

The design philosophy of building an authentication service on top of a secure time service is itself questionable. That is, it makes little sense to build an authentication system assuming an already-authenticated underlying system. Furthermore, while spoofing an unauthenticated time service may be a difficult programming task, it is not cryptographically difficult.[5] As Kerberos is proposed for more varied environments, its dependence on a secure time service is more problematic and must be stressed.

As an alternative, we propose the use of a challenge/response authentication mechanism. As is done today, the client would present a ticket, though without an authenticator. The server would respond with a nonce identifier encrypted with the session key $K_{c,s}$; the client would respond with some function of that identifier, thereby proving that it possesses the session key.

Such an implementation is not without its costs, of course. An extra pair of messages must be exchanged each time a ticket is used, which rules out the possibility of authenticated datagrams. More seriously, all servers must then retain state to complete the authentication process. While not a problem for TCP-based servers, this may require substantial modification to UDP-based query servers. This change is quite comparable in magnitude to the change needed to store live authenticators.

There is a signficant philosophical difference between the two techniques, however. In the current Kerberos implementation, with its assumptions about the network environment, retained state is only necessary to enhance security. The challenge/response scheme, on the other hand, guarantees security independent of the environment, but requires retained state to function at all.

4.3 Password-Guessing Attacks

A second major class of attack on the Kerberos protocols involves an intruder recording login dialogs in order to mount a password-guessing assault. When a user requests $T_{c,tgs}$ (the ticket-granting ticket), the answer is returned encrypted with K_c, a key derived by a publicly-known algorithm from the user's password. A guess at the user's password can be confirmed by calculating K_c and using it to decrypt a recorded T_{tgs} response. An intruder who has recorded many such login dialogs has good odds of finding several new passwords; empirically, users do not pick good passwords unless forced to.[Morr79, Gram84, Stol88]

We propose the use of exponential key exchange[Diff76] to provide an additional layer of encryption. Without describing the algorithm in detail, it involves the two parties exchanging numbers that each can use to compute a secret key. An outsider, not knowing how the numbers were calculated, cannot easily derive the key.

Such a use of exponential key exchange would prevent a passive wiretapper from accumulating the network equivalent of `/etc/passwd`. While exponential key exchange is normally vulnerable to active wiretaps, such attacks are comparatively rare, especially if dedicated network routers are used.

5. In some environments, programming is not even necessary. Low-powered fake WWV transmitters are not hard to build, and, if properly located, could easily block out the legitimate signal.

Using exponential key exchange has its costs. LaMacchia and Odlyzko[LaMa] have demonstrated that exchanging small numbers is quite insecure, while using large ones is expensive in computation time. Additionally, we have added extra messages to the login dialog, and imposed the requirement for considerable extra state in the server. Given the trend towards hiding even encrypted passwords on UNIX systems, and given estimates that half of all logins at MIT are used within a two-week period, the investment may be justifiable. Perhaps the best solution is to support this feature as a domain-specific option.

Even exponential key exchange will not prevent all password-guessing attacks. Depending on how carefully the Kerberos logs are analyzed, an intruder need not even eavesdrop. Requests for tickets are not themselves encrypted; an attacker could simply request ticket-granting tickets for many different users. An enhancement to the server, to limit the rate of requests from a single source, may be useful. Alternatively, some portion of the initial ticket request may be encrypted with K_c, preventing this sort of attack completely.

An alternative approach is a protocol described by Lomas, Gong, Saltzer, and Needham.[Loma89] They present a dialog with a server that does not expose the user to password-guessing attacks. However, their protocol relies on public-key cryptography, an approach explicitly rejected for Kerberos.

4.4 Spoofing Login

In a workstation environment, it is quite simple for an intruder to replace the `login` command with a version that records users' passwords before employing them in the Kerberos dialog. Such an attack negates one of Kerberos's primary advantages, that passwords are never transmitted in cleartext over a network. While this problem is not restricted to Kerberos environments, the Kerberos protocol makes it difficult to employ the standard countermeasure: one-time passwords.

A typical one-time password scheme employs a secret key shared between a server and some device in the user's possession. The server picks a random number and transmits it to the user. Both the server and the user (with the aid of the device) encrypt this number using the secret key; the result is transmitted back to the server. If the two computed values match, the user is assumed to possess the appropriate key.

Kerberos makes no provision for such a challenge/response dialog at login time. The server's response to the login request is always encrypted with K_c, a key derived from the user's password. Unless a ''smart card'' is employed that understands the entire Kerberos protocol, this precludes any use of one-time passwords.

An alternative (first suggested to us by T.H. Foregger) requires that the server pick a random number R, and use K_c to encrypt R. This value $\{R\}K_c$, rather than K_c, would be used to encrypt the server's response. R would be transmitted in the clear to the user. If a hand-held authenticator was in use, the user would employ it to calculate $\{R\}K_c$; otherwise, the login program would do it.

Several objections may be raised to this scheme. First, hand-held authenticators are often thought to be inconvenient. This is true; however, they offer a substantial increase in security in high-threat environments. If they are not used, the cost of our scheme is quite low, simply one extra encryption on each end.

A second, more cogent, objection is that if the client's workstation cannot be trusted with a user's password, it cannot be trusted with session keys provided by Kerberos. This is, to some extent, a valid criticism, though we believe that compromise of the login password is much more serious than the capture of a few limited-lifetime session keys. This problem cannot be solved without the use of special-purpose hardware, a subject we shall return to below.

Finally, it has been pointed out that a user can always supply a known-clean boot device, or boot via the network. The former we regard as improbable in practice unless removable media are employed; the latter is insecure because the boot protocols are unauthenticated.

4.5 Chosen Plaintext Attacks

According to the description in the Version 5 draft,[Kohl89] servers using the KRB-PRIV format are susceptible to a chosen plaintext attack. (Mail and file servers are examples of servers susceptible to such attacks.) Specifically, the encrypted portion of messages of this type have the form

$$X = (DATA,\ timestamp + direction,\ hostaddress,\ PAD)$$

Since cipher-block chaining has the property that prefixes of encryptions are encryptions of prefixes, if *DATA* has the form

$$(AUTHENTICATOR,\ CHECKSUM,\ REMAINDER)$$

then a prefix of the encryption of X with the session key is the encryption of (*AUTHENTICATOR*, *CHECKSUM*), and can be used to spoof an entire session with the server.

It may be argued that most servers are not susceptible to chosen plaintext attacks. Given that there are easy counters to this attack, it seems foolish to advocate a general format for private servers that does not also protect against it.

It should be noted that the simple attack above does not work against Kerberos Version 4, in which the encrypted portion of the KRB_PRIV message is of the form

$$(length(DATA),\ DATA,\ msectime,\ hostaddress,\ timestamp + direction,\ PAD)$$

as the leading $length(DATA)$ field disrupts the prefix-based attack. We leave it to the reader to discover a more complicated chosen ciphertext attack against this format, even allowing for the fact that Version 4 uses the nonstandard PCBC mode of encryption. (Hint: assume the initial vector is fixed and public.) However, it is interesting to note that the order of concatenation of message fields can have security-critical implications. We return to this question in the later section on message encoding.

4.6 Session Keys

The term ‘‘session key’’ is a misnomer in the Kerberos protocol. This key is contained in the service ticket and is used in the multiple sessions between the client and server that use that ticket. Thus, it is more properly called a ‘‘multi-session key’’. Making this point explicit leads naturally to the suggestion that true session keys be negotiated as part of the Kerberos protocol. This limits the exposure to cryptanalysis[Kahn67, Beke82, Deav85] of the multi-session key contained in the ticket, and precludes attacks which substitute messages from one session in another. (The chosen-plaintext attack of the previous section is one such example.) The session key could be generated by the server or could be computed as a session-specific function of the multi-session key.

4.7 Message Encoding

The most simple analysis of the security of the Kerberos protocols should check that there is no possibility of ambiguity between messages sent in different contexts. That is, a ticket should never be interpretable as an authenticator, or vice versa. Such an analysis depends on redundancy in the pre-encryption binary encodings of each of the ticket and authenticator information. Currently, that analysis must be repeated with every modification to the protocol. This repetitive and often intricate analysis would be unnecessary if standard encodings (such as ASN.1)[ASN1, BER] were used. These encodings should include the overall message type (such as KRB_TGS_REP or KRB_PRIV). Together with reasonable assumptions about the encryption layer (see the next section), such an encoding scheme would greatly simplify the protocol validation process, particularly as the protocol is modified or extended.

The use of ASN.1 encodings is being considered for other reasons in Version 5. We reinforce here that there are design principles other than standards compatibility that motivate such a change.

4.8 The Encryption Layer

Version 4 of Kerberos uses the nonstandard PCBC mode of encryption, *propagating cipher block chaining*, in which plaintext block $i+1$ is exclusive-or'ed with both the plaintext and ciphertext of block i before encryption. This mode was observed to have poor propagation properties that permit message-stream modification: specifically, if two blocks of ciphertext are interchanged, only the corresponding blocks are garbled on decryption. Version 5 replaces PCBC mode with the standard CBC mode, *cipher block chaining*, which exclusive-or's just the ciphertext of block i with the plaintext of block $i+1$ before encryption. A checksum — as of Draft 2, the exact form had not been determined — is used to detect message modification. In order to ensure that duplicate messages have different encryptions, random initial ''confounders'' are added to some message formats. In addition, Version 5 supports alternative encryption algorithms as options.

Both the confounder and checksum mechanisms are meant to augment the security of CBC encryption. They belong in a separate encryption layer, not at the level of the Kerberos protocols themselves. Further, the confounder mechanism should be replaced by using the standard initial vector mechanism of cipher-block chaining.

To prevent message-stream modification during authenticated or private sessions, Version 5 uses a timestamp field to prevent entire encrypted messages from being replayed. This is another concern more properly delegated to the encryption layer, where chaining across the packets of the entire session is the more standard mechanism. (Such chaining avoids both the dependence on a clock and the need to cache recent timestamps.)

Separating the Kerberos protocols from the details of encryption would facilitate both validation of the security of the Kerberos protocols, and implementations and validations involving alternative cryptosystems. Too much focus on mechanism, while endemic to cryptographic protocol design, leads away from the need to state the basic properties required of the encryption layer. We would suggest the following adversarial analysis as the starting point for such a specification: allow an adversary to submit, one after the other, any number of messages for encryption under an unknown key K. The adversary also has the ability to take prefixes and suffixes of known messages, exclusive-or known messages, and encrypt or decrypt with known keys. At the end of this process, the adversary should not be able to produce any encrypted messages other than those specifically submitted for encryption. Such an analysis would preclude encryption schemes susceptible to simple chosen-plaintext attacks, as described in a previous section.

Given the intractability of reasoning about DES, or of proving complexity properties of any cryptosystem with bounded key size, such analyses will be no guarantee of overall security. But they can be used to preclude the existence of trivial cut-and-paste attacks.[DeMi83, Moor88]

4.9 The Scope of Tickets

What should the scope of a ticket be, both in time and space? Put another way, how long should tickets last? A short lifetime is inconvenient, especially for background jobs. On the other hand, long-lived tickets are vulnerable to capture. There is no single right answer, but Kerberos Version 4 is clearly wrong. By this we mean that its facilities are insufficient, and do not provide either users or administrators with the necessary flexibility.

Similarly, on which machines should a ticket be valid? A ticket obtained on one machine must be transferable to others in order to permit third-party file copies, use of network resources from a machine remote to the original one, etc. But to accept such tickets implies trust of the remote machine, and in particular trust that the remote machine is immune to some of the attacks mentioned here.

Kerberos (Version 4) takes a simple approach. Maximum ticket lifetime is settable on a per-user basis (strictly speaking, a per-principal basis); however, it can never exceed 5×255 minutes, or about 21 hours. Tickets are not renewable.

Its provisions for forwarding is similarly simple, and similarly restrictive: ticket-forwarding is not permitted. That is, tickets may exist only on the machine on which one's password was entered, either via `login` or by explicit use of the `kinit` command. The remedy — manually using `kinit` on all remote machines — is inadequate as it requires entry of a password; this password is transmitted unencrypted across the network.

One manual mechanism was grafted onto the side of the Kerberos Version 4 protocol to address this problem. It involves a special remote ticket daemon that must be run on each remote machine. A user who wished to transfer tickets to such a machine would enter the `rkinit` command; this would request tickets for that machine, but with a password being entered only on the user's local machine.

Both of these problems are largely solved in the proposed Version 5 protocol. Tickets may be marked as forwardable, renewable, or even post-dated. Actually using any of these features requires an extra interaction with Kerberos to validate the tickets; thus, the server could maintain a ''hit list'' of tickets known to be compromised.

A ticket's extent is limited in other ways, in particular to the *realm* of its Kerberos server. Interrealm authentication is possible if the two Kerberos realms share a private key; unfortunately, in Version 4 this implies the need for n^2 keys if n different realms wish to share authentication. Version 5 permits a hierarchy of realms, and passes the complete authentication path on to the destination Kerberos; this allows it to decide if all parties along the way are trustworthy.

5. ON THE NEED FOR SPECIAL HARDWARE

One of the major reasons we question the suitability of Kerberos for multi-user hosts is the need for plaintext key storage. What if the host were equipped with an attached cryptographic unit? We consider the design parameters for such a box.

The primary goal is to perform cryptographic operations without exposing any keys to compromise. These operations must include validating tickets presented by remote users, creating requests for both ticket-granting tickets and application tickets, and encrypting and decrypting conversations. Consequently, there must be secure storage for a large number of keys, and the operating system must be able to select which key should be used for which function.

The next question, of course, is how keys are entered into the secure storage area. If tickets are decrypted by the encryption box but transferred to the host's memory for analysis, the embedded session key is exposed. Indeed, such attacks have been used on conventional passwords on UNIX systems where `/dev/kmem` (kernel memory) was generally readable. Therefore, we conclude that the encryption box itself must understand the Kerberos protocols; nothing less will guarantee the security of the stored keys.

Entry of user keys is more problematic, since they must travel through the host. Unless user terminals are connected directly to the encryption unit, there is little choice. Storing them off the host, though, is a significant help, as the period of exposure is then minimized. Host-owned keys — service keys, or the keys that `root` would use to do NFS mounts — should be loaded via a Kerberos-authenticated service resident in the encryption unit.

We must now ensure that the protocol itself does not provide a mechanism to obtain keys. Looking at the message definitions, we see that only session keys are ever sent, and these are always sent encrypted. Furthermore, user machines never generate any such messages; they merely forward them. Thus, the box need not have the ability to transmit a key, thereby providing us with a very high level of assurance that it will not do so.

If an encryption box is used for the Kerberos server itself, the problem is somewhat more complex. There are two places where keys are transmitted. First, when a ticket is granted, the ticket itself contains a session key, and a copy of that session key is sent back encrypted in the client's ticket-granting session key. Second, during the initial dialog with Kerberos, the ticket-granting session key must be sent out, encrypted in the client's password key. Note, though, that permanent keys are *never* sent; again, this assures us that the encryption box will not give away keys. Furthermore, since these session keys are intended to be random, we can buy ourselves a great deal of security by including a hardware random number generator on-board.

We are not too concerned about having to load client and server keys onto the board. This operation is done only by the Kerberos master server, for which strong physical security must be assumed in any event. It is possible that such an encryption unit can be made sufficiently tamper-resistant that even workstations can use them; certainly, there are commercial cryptographic devices that claim such strengths.

One major objection to this entire scheme is that ultimately, the encryption box is controlled by the host computer. Thus, if `root` is compromised, the host could instruct the box to create bogus tickets. Such concerns are certainly valid. However, as noted above, we consider such temporary breaches of security to be far less serious than the compromise of a key. Furthermore, using a separate unit allows us to create untamperable logs, etc.

It is also desirable to prevent misuse of keys. For example, we do not want the login key used to decrypt the arbitrary block of text that just happens to be the ticket-granting ticket. Accordingly, keys should be tagged with their purpose. A login key should be used only to decrypt the ticket-granting ticket; the key associated with it should be used only for obtaining service tickets, etc. Since the encryption box is performing all of the key management, this is not a difficult problem.

6. RECOMMENDED CHANGES TO THE KERBEROS PROTOCOL

Below, we list our recommended changes to the Kerberos protocol. Our ranking is governed by our estimate of the likelihood and consequences of the attack, balanced against the difficulty of implementing the modification.

a. A challenge/response protocol should be offered as an optional alternative to time-based authentication.

b. Use a standard message encoding, such as ASN.1, which includes identification of the message type within the encrypted data.

c. Alter the basic login protocol to allow for handheld authenticators, in which $\{R\}KC$, for a random R, is used to encrypt the server's reply to the user, in place of the key KC obtained from the user password. This allows the login procedure to prompt the user with R, who obtains $\{R\}KC$ from the handheld device and returns that value instead of the password itself.

d. Mechanisms such as random initial vectors (in place of confounders), block chaining and message authentication codes should be left to a separate encryption layer, whose information-hiding requirements are clearly explicated. Specific mechanisms based on DES should be validated and implemented.

e. The client/server protocol should be modified so that the multi-session key is used to negotiate a true session key, which is then used to protect the remainder of the session.

f. Support for special-purpose hardware should be added.

g. Support for optional extensions should be included. In particular, an option to protect against dictionary attacks on `/etc/passwd` may be a desirable extension.

REFERENCES

[ASN1] ‘‘Information Processing Systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1),’’ International Standard 8824 (1987). International Organization for Standardization and International Electrotechnical Committee

[BER] ‘‘Information Processing Systems — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1),’’ International Standard 8825 (1987). International Organization for Standardization and International Electrotechnical Committee

[Beke82] H. Beker and F. Piper, *Cipher Systems,* John Wiley & Sons (1982).

[Davi89] D. Davis and R. Swick, *Workstation Services and Kerberos Authentication at Project Athena,* MIT Project Athena (March 3, 1989).

[Deav85] C.A. Deavours and L. Kruh, *Machine Cryptography and Modern Cryptanalysis,* Artech House (1985).

[DeMi83] R. DeMillo and M. Merritt, ‘‘Protocols for Data Security,’’ *Computer* **16**(2), pp. 39-50 (February 1983).

[Diff76] W. Diffie and M.E. Hellman, ‘‘New Directions in Cryptography,’’ *IEEE Transactions on Information Theory* **6**, pp. 644-654 (November, 1976).

[Gram84] F.T. Grampp and R.H. Morris, ‘‘UNIX Operating System Security,’’ *AT&T Bell Laboratories Technical Journal* **63**(8, Part 2), pp. 1649-1672, A&T (October, 1984).

[Kahn67] D. Kahn, *Codebreakers: The Story of Secret Writing,* Macmillan (1967).

[Kohl89] J. Kohl, B. Clifford Neuman, and J. Steiner, *The Kerberos Network Authentication Service,* MIT Project Athena (November 6, 1989). Version 5, Draft 2

[LaMa] B.A. LaMacchia and A.M. Odlyzko, *Computation of Discrete Logarithms in Prime Fields*, (Manuscript in preparation).

[Loma89] T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham, ‘‘Reducing Risks from Poorly Chosen Keys,’’ *Operating Systems Review* **23**(5), pp. 14-18, ACM (December 1989).

[Mill88] D.L. Mills, ‘‘Network Time Protocol,’’ RFC 1059 (July 1988).

[Mill89] D.L. Mills, ‘‘Network Time Protocol,’’ RFC 1119 (September 1989).

[Moor88] J.H. Moore, ‘‘Protocol Failures in Cryptosystems,’’ *Proc. IEEE* **76**(5), pp. 594-602 (May 1988).

[Morr79] R. Morris and K. Thompson., ‘‘UNIX Password Security,’’ *Communications of the ACM* **22**(11), p. 594 (November 1979).

[Post80] J.B. Postel, ‘‘User Datagram Protocol.,’’ RFC 768 (August 28, 1980).

[Post81] J.B. Postel, ‘‘Transmission Control Protocol.,’’ RFC 793 (September 1981).

[Post83] J.B. Postel and K. Harrenstien, ‘‘Time Protocol.,’’ RFC 868 (May 1983).

[Stei88] J. Steiner, C. Neuman, and J.I. Schiller, ‘‘Kerberos: An Authentication Service for Open Network Systems,’’ in *Proc. Winter USENIX Conference*, Dallas (1988).

[Stol88] C. Stoll, ‘‘Stalking the Wiley Hacker,’’ *Communications of the ACM* **31**(5), p. 484 (May 1988).