Secure Remote Login

Lawrence Brown
lpb@cs.adfa.oz.au

Department of Computer Science
Centre for Computer Security Research
University College, UNSW, Australian Defence Force Academy
Canberra ACT 2600. Australia.

Abstract

In this paper I discuss some alternative approaches to providing security in remote terminal sessions. Since remote terminal sessions mean that information is carried over a network that may not be trusted, there is a need for mechanisms to protect the security of information exchanged during the session. The protocols implemented will need to address the issues of authentication of the parties, negotiation of a suitable cipher to use, and the exchange of keys for that cipher. In this paper I examine each of these issues. I then describe a prototype implementation incorporating some of these features in the **seclog** program, and I finish with a description of work in progress on extending the widely used **telnet** protocol to incorporate some of these security features.

1. Introduction

With the growth of wide area networks linking together existing local area networks, there are increasing security problems with the existing network utilities traditionally supplied for such LANs. These utilities have assumed that the LAN was a secure communications service (even though it was recognised fairly early that it was not necessarily so), and that hence additional security services for these utilities were not supplied. However, a wide-area internet certainly cannot be assumed to be secure[1]. Thus there is a desire to provide utilities with additional security. Some of the issues involved in providing such security are canvassed in [WoK85 pp 189-198] [HeR89] and [DaP89].

In this paper, I start by identifying some of the issues involved with security when using a remote terminal session over a network. I then identify the areas of protocol extension needed to prevent violating the security of information exchanged during such a session, and where such extensions might be incorporated intot he existing protocols. These protocols extensions will need to address the issues of authentication of the parties, negotiation of a suitable cipher to use, and the exchange of keys for that cipher. Then I describe a prototype which incorporated some of these extensions into the **seclog** program. This involved writing a pair of programs external to the existing rlogin program, which provided the necessary security by utilising the existing rlogin protocol unchanged. This resulted in a number of problems in practical implementation, which indicated that these extensions really need to be incorporated into the underlying protocols. I finish with a description of work in progress on extend the option mechanism of the widely used **telnet** protocol to incorporate some of these security

[1] For a fascinating account of the 1986 Internet breakin, see [Sto88], and for details on other breakins see [Rei87] [Spa88].

extensions.

2. Login Over a Network - Some Issues

2.1. Identity of the User

When a user logs in to a host computer, they provide a username and password to identify themselves to the computer. On a hardwired line this is assumed to be secure, as it is difficult for someone to tap such a line and intercept the username password pair. However when logging in over a network, anyone with access to a workstation and packet monitoring software could intercept the pair, and then subsequently claim to be the original user (see Fig 1). In either case it is assumed that the password is well chosen, so that it cannot be guessed [MoT79] [Spa88 Appendix A].

To provide for authentication of the user, the information sent over the network should be insufficient to allow an intruder to impersonate the user without additional knowledge. A third party, who may or may not be trusted could be involved in this negotiation. The parties would exchange information until they were convinced of each others identity, without releasing unnecessary information to an eavesdropper. There are a number of ways of doing this.

Fig 1 - Identity of User Uncertain

- Challenge - Response

  This involves the host computer sending a random number, a **challenge** vector, to the user. The user encrypts this using a **private** key known only to both parties, with an agreed cipher, and sends the encrypted **response** vector to the host. The host encrypts the **challenge** vector using the user's **private** key, and compares it with the **response** returned. The user is accepted only if they are identical.

  Since any intruder monitoring the network will see a different challenge and response each time, they will not be able to produce a correct response, unless they have obtained the user's key by some other means (which is a separate problem). This technique assumes that the user and remote host have already arranged a private key prior to wanting to establish a remote login session. Since, in this application, a user must have negotiated with the remote site to obtain a usercode and conventional password, obtaining a network private key simply becomes another piece of information exchanged during this negotiation. An issue that needs to be addressed in a practical implementation of this scheme is which cipher to use to build the response. Any of the modern block ciphers detailed later would suffice, so the choice may either be fixed, or made part of the negotiation in the protocol.

- Kerberos

  This may be used if both of the parties who wish to engage in a remote terminal session participate in a common Kerberos authentication domain (by each sharing keys with trusted key servers, which in turn are part of the same hierarchy of servers, and thus can exchange information). In this case, the Kerberos authentication system may be used to create a special signed message, called a **ticket**, which validates the identity of each of the parties, and in addition may include a suitable session key for use later. In essence this is deferring the security issue to another domain, and indeed the Kerberos system incorporates some of the ideas discussed here. This method relies on the use of a trusted third party to assist the authentication process. For further details on the Kerberos scheme see [SNS88] and [Koh90]. For a critique of the system see [BeM90].

  A similar mechanism is also provided in the Amoeba distributed operating system. For details of this see [MRT90] [TMR86].

- Digital Signature

  This method uses a public-key authentication scheme [SeP89] to identify the user. To use it, the user signs an identity message with the private half of the key and sends this to the remote host. There it is verified by using the public half of the key to confirm that the identity message is unchanged. Since only the user knows the private half of the key, only they could have signed the identity message. However, there is still a problem in obtaining the correct public half of the key. If it is obtained from some remote database, an intruder could intercept the request, and return a fraudulant public-key claiming to be the desired recipient. The intruder could then claim to be the recipient. This problem is solved by providing the public key signed by a public key centre (or possibly a hierarchy of signed keys), whose public key is well known. This can be supplied by the user as part of the identity message which removes the need for a network key server to be accessed on every remote terminal session request. Whilst this method has a number of nice properties, including no need for prior key exchange, it is computationally slow. However if a public key scheme is to be used for encryption, then this would be the obvious counterpart. Candidates for practical implementation include RSA [RSA78] or El Gamal [ElG85] [McC86].

- Zero Knowledge Proof of Identity

  This technique provides a way of authenticating a user, by using a series of challenges and responses to convince the remote site that the user is who they say they are. However it does not require the prior arrangement of a key. In this method, the user sends a certificate (Ident,j) originally obtained from a public key centre, to the remote host. The host then interacts with user to prove ownership of the certificate [FFS88] [SeP89]. This method is much faster than a digital signature, though it involves a dialog between the user and the host. In the longer term, it will probably be the method of choice.

2.2. Privacy of Session

Having authenticated the user, the next issue to address is the provision of privacy for the user (see Fig 2).

This implies a need to encrypt the data transferred during the session. However, a choice has to be made as to which form of cipher is to be used to provide privacy. A Public-Key scheme has the nice property that the keys need not be exchanged prior to establishing a session, but current implementations are too slow. The alternative is to use a private-key scheme. These are considerably faster, but require the keys to have been exchanged in some secure fashion, either prior to the session, or via a public key exchange scheme. It is anticipated that in the near future private-key schemes will be used. As computer power continues to increase though, public-key schemes will become more desirable and will need also to be included.

Fig 2 - Privacy of Session

Having decided to use a private key scheme, there is still a range of ciphers to chose from, trading off speed for security. The choices include:

- Simple Substitution Cipher

  This is very fast, very simple to implement, but is very insecure. Guessing as many characters as there are in the key is sufficient to crack the scheme. However, if all that is required is to defeat casual scanning (or an automatic word scanner), then it may well suffice.

- Stream Cipher

  Stream Ciphers are usually based on a pseudo-random number generator. They are reasonably fast, and given a suitable choice of generator (such as a non-linear feedback shift register, or multiplexed linear feedback shift registers), have a moderate to high level of security. They can be cracked by guessing a sufficiently large number of characters, but the number needed could be very large. Another possible precaution is to change keys before this number of characters is sent (using for example the challenge-response key again, and encrypting the previous session key again every period).

- Block Cipher

  The cipher would be used in a stream mode. Any of the modern block ciphers may be used, eg DES [ASA85] LOKI [BPS90] FEAL8 [ShM88] [Miy90]. These schemes all have a very high level of security, but are probably the slowest of the private-key alternatives. Suitable modes include either Cipher Feedback (CFB), or Output Feedback (OFB) modes (see Fig 3). If the CFB mode is chosen, it uses the message contents in forming the new key (which can slow down parallel key stream generation), but which leads to different key streams for different messages. If the OFB mode is used, the key stream generated is independent of the message content (and thus the key stream can be precomputed). However OFB should not be used to encrypt a number of different messages with the same keystream, as this creates a security hole. However, if different keys are used, it is equivalent in security to CFB, with the advantage of being able to precompute the keystream if desired.

Fig 3 - Stream Modes for Block Ciphers

2.3. Exchange of Keys

Before a cipher can be used to encrypt data, it is necessary to mutually agree on a suitable key. Such a key may be simply agreed on outside of the protocol altogether (it may simply be another password), may be exchanged during, or derived from the authentication stage, or may be negotiated by a completely separate phase. In more detail these alternatives are:

- Keys Exchanged Outside the Protocol

  In this case the user is given a key by means completely independent to the remote login negotiation. Typically this would be an additional password supplied to the users by some secure physical means. This key could be just for the encryption phase, but more typically would also be used in the authentication phase, and then a session key would be derived to use in the encryption. This then becomes the next alternative.

- Key Derived from Authentication Process

  In this approach, the key used to encrypt data is derived from information exchanged during the authentication negotiation. For example, if a **challenge-response** scheme is being used, then a suitable key may be derived by encrypting the **response vector** to form the session key. If **Kerberos** or a like scheme is used, then the session key may well be incorporated into the **ticket** exchanged, along with verification of the users identity. Similarly, if a digital signature is being used along with public-key encryption of the data, then the signature may include a certificate, signed by a public notary, identifying the user and specifying the public key to be used.

- Public-Key Exchange

  The final alternative is to use a separate phase of the negotiation to exchange a key using a public-key exchange protocol. In this case the exact protocol to be used would need to be negotiated, and then the exchange could take place. Possible schemes here include using RSA [RSA78], appropriate if RSA is used to provide either authentication or secrecy or both; or Diffie-Hellman [DiH76], appropriate if El Gamal is used in either the authentication or secrecy stages.

The choice of which of these alternatives, and how many if there is scope for negotiation, will depend on the actual protocol being modified to incorporate these additional security features.

3. Possible Implementation Alternatives

3.1. Where to Add Security?

Since there are a number of existing protocols for remote terminal sessions, eg rlogin or telnet, it would be preferable to adapt these to provide the security features detailed earlier, rather than write a new protocol. The issue is how to adapt these existing tools. There are two major options, as shown in Fig 4.

(A) Integrated into the Utilities

Here, the programs are modified to include the required security features. This has the advantage of being able to properly merge the extensions with the existing protocols. However, access is required to the source to achieve this on all systems where it is required. In the longer term, this is the option of choice, and this paper will conclude with the state of negotiations to add security extensions to the Telnet protocol.

(B) Using Additional Processes

Here, some filters are added, in traditional Unix fashion, around the existing utilities. This is much easier to do, since there are no changes to the existing tools, but there are several disadvantages. It adds more processes and thus context switches, it blocks user interaction with the client program using escapes, and it requires a transparent (8-bit) path. Nonetheless, this was the option chosen for the prototype implementation, since it could be used to verify the functionality of the security aspects of the protocol, even though there were other problems with the scheme.

Fig 4 - Adding a Security Utility

3.2. Phases of the Protocol

Given the discussion in the previous section, the security additions will need to incorporate the following phases of negotiation:

• Authentication Phase

this includes negotiation of a particular type of authentication protocol to use (eg Kerberos, challenge-response etc) and the associated exchanges of information (eg Kerberos ticket, username challenge & response vectors etc).

• Choice of Cipher Phase

this includes the negotiation of which cipher to use to provide encryption (eg DES, LOKI, FEAL, RSA, simple substitution cipher etc), the mode of use of this cipher (CFB, OFB), and implicitly a size of key to use (eg 64-bits for DES or LOKI, perhaps 128-bits for an extended DES like scheme, 512-bits or more for RSA etc).

• Choice of Key Phase
    this will determine how the session key to be used in the chosen cipher is to be established. It may be derivable from the previous authentication exchange (eg be included in the Kerberos ticket, be derived from the response vector etc), or it may be exchanged via a further negotiation (eg using Diffie-Hellman key exchange, RSA key exchange etc).

Once all of these have been negotiated, then it is possible to enable and disable encryption of information on the link, via suitable handshaking sequences.

Additional aspects that need to be addressed as part of these extensions include, how much of the data should be encrypted, whether both input and output streams are encrypted, or perhaps only (say input whilst reading a password), and how to reestablish synchronization of the ciphers, should a transmisison error occur. Again, the choices for these will depend on the actual protocol being modified.

4. SECLOG - A Secure Remote Login Prototype

4.1. Overview

SECLOG is a prototype implementation of some utilities to provide a basic secure login service. Structurally, it uses extra processes around rlogin (option B). Rlogin rather than telnet, was chosen as the basic remote terminal service because it provides a transparent (8-bit) mode (using the undocumented -8 flag). This made implementation a little easier. In the SECLOG prototype, the following choices were made with respect to the phases of negotiation. A challenge-response authentication scheme, using the DES cipher was chosen. Keys for this were to be derived external to the protocol (in the nature of additional passwords). The session key to be used to encrypt session data was derived from the response vector. A negotiation of the cipher to use was provided. Choices ranged from simple substitution to LOKI or DES block ciphers. In more detail, SECLOG performs the following operations:

automatic login to a secure userid
challenge/response to verify user
derive session key from response
negotiates encryption cipher to use
perform encryption of session data

It does not need to run as a a privileged (setuid) program, though it does require a small privileged utility to cleanup utmp/wtmp and reset the terminal permissions. It assumes that a network key (here it is an 8-char key used in the chaalenge-response stage) has been arranged previously.

The SECLOG processes are structured as shown in Fig 5. The client program **seclog** is run by the user to establish a secure remote terminal session. It then runs rlogin, which establishes a link to rlogind on the remote host. The **seclog** client then logs into the secure user on the remote system, which in turns runs the **seclogd** server. These then negotiate the challenge/response and cipher to use, and if successful, the server runs login to actually login the user and start their shell. For the remainder of the session, the client and server encrypt all information being transferred between them. Further details of the SECLOG prototype may be found in the earlier version of this paper [Bro90].

4.2. SECLOG - Problems Observed

Whilst writing and debugging **seclog**, a number of problems were found, including:

- The handling of prompts and replies during login. It was very easy to get out of sync between the client and server during the automated login.
- Sync of encrypted data during session. This was a major problem. Apart from ensuring that the cipher algorithms themselves stay in sync, a more serious problem was the use of stderr by programs inside the encrypted data path, especially messages from **rlogin** and the **seclogd** server.
- Problems with the handling of utmp/wtmp entries, and resetting the terminal permissions for logged in users.

It is clear that overcoming these problems will require an alternative approach, one in which the security extensions are much more closely integrated with the existing protocol.

–

Fig 5 - SECLOG Processes

5. STELNET - Security Extensions to Telnet

5.1. Why Telnet

Having gained some experience with SECLOG it was apparent that the security extensions should be integrated into an existing remote terminal program. The telnet and telnetd programs were chosen for this, both because of their wide use round the world as part of the TCP/IP protocols, their implementation on a range of hosts and operating systems, availability of source, and finally, a readily extensible option mechanism which lends itself to such additions. Implementing the extension in this manner will bypass many of the problems found in the prototype implementation, due to the much closer merging of the features with the existing protocols.

The telnet protocol and its option mechanisms are well documented in RFCs [PoR83a] [PoR83b] [Bra89]. The option mechanism for telnet is easily extended simply by defining new option numbers. If a telnet program does not know about an option, it simply declines to negotiate about it further by using the standard DONT/WONT protocol supported for all options. If both parties do understand and wish to use, they agree to discuss the option using DO/WILL, and may then actually perform the negotiations using sub-options specific to each telnet option.

5.2. Proposed Security Options for Telnet

I have drafted some proposed additional options to provide network security in the Telnet options. Official telnet extensions need to be vetted and approved by the Telnet Working Group of the Internet Engineering Task Force, in the US. To date, I have submitted these proposals for adding security options to telnet to them. These, along with some alternative proposals, are currently being considered. At the time of writing a firm descision has yet to be made. The result will almost certainly incorporate aspects from both of the proposals being considered. It appears likely that the final proposal will have the following form. There will be two additional options: the **Telnet Authentication** option, and the **Telnet Data Encryption Option**. It is likely that the exchange between the two parties using these options will run as shown in Fig 6 and Fig 7.

Client		Server	Comments
<---	DO / DONT		agree to discuss authentication
	WILL / WONT	--->	option (or not)
<---	SEND <auth-list>		server tells client the list of authentication protocols it supports
	IS <auth-type>, <arg>	--->	client tells server which protocol it will use, and supplies the info
<---	CHALLENGE <arg>		servers sends challenge to client (optional depending on auth-type)
	RESPONSE <arg>	--->	client replies to challenge

Fig 6 - Telnet Authentication Option

In Fig 6, the parameters referenced are as follows:

<auth-list>
: refers to a list of possible authentication protocols supported (eg challenge-response, Kerberos, El Gamal digital signature, etc).

<auth-type>, <arg>
: names a particular authentication protocol, and supplies additional information as appropriate. If the auth-type is challenge-response, this would be the name of the user. If Kerberos, it would be the Kerberos ticket.

<arg>
: indicates additional information as required by the protocol at that point.

The authentication types are represented as a list of known integer values. These values would be documented in the "Assigned Numbers RFC".

The above exchange enables the client to authentiate their identity with the server. Should two-way authentication be required, then the exchange may be repeated with roles reversed. The proposed protocol can accomodate advances in encryption technology, in that new authentication protocols can easily be added by extending the list of known protocols.

Client		Server	Comments
<---	DO / DONT		Agree to discuss data encryption
	WILL / WONT	--->	option (or not)
<---	SUPPORT <cipher-list>		server sends client list of ciphers supported (including option of NONE)
	USE <cipher>	--->	client nominates which cipher to use
<---	KEY EXCH <key-list>		server sends client the list of key exchange protocols supported (includes use of default key from auth negotiation or explicit list of protocols)
	KEY1 <key-type arg>	--->	client nominates which key exchange protocol to use and optionally provides some information for it (eg part of Diffie-Hellman exchange).
<---	KEY2 <key-type> <arg>		server optionally responds with its half of the key exchange protocol information (eg part of Diffie-Hellman)
<---	REQ_START CIPHER		Servers asks client to encrypt
	START CIPHER	--->	Client will encrypt session data
<---	REQ_STOP CIPHER		Servers asks client to stop encrypting
	STOP CIPHER	--->	Client will stop encrypting session data

Fig 7 - Telnet Data Encryption Option

In Fig 7, the parameters referenced are as follows:

<cipher-list>
: refers to a list of ciphers and associated modes supported (eg LOKI in OFB mode, LOKI in CFB mode, DES in OFB mode, simple substitution etc).

<cipher>
: names a particular cipher and mode chosen from the list, to be used.

<key-list>
: refers to a list of possible session key determination protocols (eg external, derive from authentication phase, use Diffie-Hellman key exchange etc).

<key-type arg>
: supplies the information needed as part of a key exchange process (for example, the components of a Diffie-Hellman key exchange).

These exchanges allow the parties to negotiate a mutually agreeable cipher and session key, and then to enable and disable encryption on each of the data streams.

Again, as in the authentication option, encryption is negotiated separately for each direction. Also extensions in encryption technology may be handled by adding new values to the list of ciphers supported.

5.3. Implementation

At present we plan to implement the extensions to telnet in the Berkeley 4.3 versions of telnet and telnetd. This will be done as soon as the extensions have been agreed upon. In the longer term, they will also be added to the NCSA version of telnet, which can be run on PCs and Macintoshs.

6. Conclusions

This paper has examined means of providing security in remote terminal sessions. Such protocols will need to address the issues of authentication of the parties, negotiation of a suitable cipher to use, and exchange of keys for that cipher. This paper examines some of the alternatives available to address each of these issues, describes a prototype implementation of some of these features in the **seclog** program, and finishes with a description of work in progress on extending the widely used **telnet** protocol to incorporate some of these security features.

Acknowledgements

To Geoff Collins, Andrzej Goscinski, Per M. Hoff, Mike and Cathy Newberry, and Chris Vance for their help and suggestions. Thankyou.

This work is supported in part by ARC grant A48830241, and Telecom Australia Research contract 7027.

Bibliography

[ASA85] ASA, "*Electronics Funds Transfer - Requirements for Interfaces, Part 5, Data Encryption Algorithm*," AS2805.5-1985, Standards Association of Australia, Sydney, Australia, 1985.

[BeM90] S. M. Bellovin and M. Merritt, "*Limitations of the Kerberos Authentication System*," submitted to Computer Communication Review, JUNE 1990.

[Bra89] "*Requirements for Internet Hosts -- Application and Support*," RFC 1123, Internet Engineering Task Force, |October.| 1989.

[Bro90] L. Brown, SECLOG - A Secure Remote Login Frontend and Shell, vol. 11, |April.| 1990.

[BPS90] L. Brown, J. Pieprzyk and J. Seberry, "LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications," in *Advances in Cryptology: Auscrypt '90*, Lecture Notes in Computer Science, no. 453, pp. 229-236, |Springer Verlag|, Berlin, 1990.

[DaP89] D. W. Davies and W. L. Price, *Security for Computer Networks*, John Wiley and Sons, New York, 1989. (2nd edn).

[DiH76] W. Diffie and M. E. Hellman, "New Directions In Cryptography," *IEEE |Trans.| on Information Theory*, vol. IT-22, no. 6, pp. 644-654, |November.| 1976.

[ElG85] T. ElGamal, "A Public Key Cryptosystem and a Signature System Based in Discrete Logarithms," *IEEE |Trans.| on Information Theory*, vol. IT-31, no. 4, pp. 469-472, |July| 1985.

[FFS88] U. Feige, A. Fiat and A. Shamir, "Zero-knowledge proofs of identity," *Journal of Cryptology*, vol. 1, no. 2, pp. 77-94, |Springer Verlag|, 1988.

[HeR89] M. Henning and A. Rohde, *UNIX File System Security*, CCSR Tutorial Series in Computer Security, no. 1, Centre for Computer Security Research, Dept Computer Science, UC ADFA, Canberra, ACT, Australia, 1989.

[Koh90] J. T. Kohl, "The Use of Encryption in Kerberos for Network Authentication," in *Advances in Cryptology - CRYPTO'89*, Lecture Notes in Computer Science, no. 435, G. Brassard (editor), pp. 35-43, |Springer Verlag|, Berlin, 1990.

[McC86] K. S. McCurley, "Digital Signatures and Public-Key Cryptosystems Based on Discrete Logarithms and Factoring," in *Crypto 86 - Abstracts and Papers*, pp. LP1.1-LP1.11, Santa Barbara, CA USA, |August.| 1986.

[Miy90] S. Miyaguchi, "The FEAL-8 cryptosystem and a call for attack," in *Advances in Cryptology - CRYPTO'89*, Lecture Notes in Computer Science, no. 435, G. Brassard (editor), pp. 624-627, |Springer Verlag|, Berlin, 1990.

[MoT79] R. Morris and K. Thompson, "Password Security: A Case History," |*Comm. of the ACM*|, vol. 22, no. 11, pp. 594-597, |November.| 1979.

[MRT90] S. J. Mullender, G. Rossum, A. S. Tanenbaum, R. Renesse and H. Staveren, "Amoeba - A Distributed Operating System for the 1990's," *IEEE Computer Magazine*, vol. 23, no. 5, pp. 44-53, |May| 1990.

[PoR83a] J. Postel and J. Reynolds, "*Telnet Protocol Specification*," RFC 854, Internet Engineering Task Force, |May| 1983.

[PoR83b] J. Postel and J. Reynolds, "*Telnet Option Specification*," RFC 855, Internet Engineering Task Force, |May| 1983.

[Rei87] B. Reid, "Reflections on Some Recent Widespread Computer Break-Ins," |*Comm. of the ACM*|, vol. 30, no. 2, pp. 103-105, |February.| 1987.

[RSA78] R. L. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," |*Comm. of the ACM*|, vol. 21, no. 2, pp. 120-126, |February.| 1978.

[SeP89] J. Seberry and J. Pieprzyk, *Cryptography: An Introduction to Computer Security*, |Prentice Hall, Englewood Cliffs, NJ|, 1989.

[ShM88] A. Shimizu and S. Miyaguchi, "Fast Data Encipherment Algorithm FEAL," *Eurocrypt 87 Abstracts* , pp. 11-14, 1988.

[Spa88] E. H. Spafford, "*The Internet Worm Program: An Analysis*," Technical Report Computer Science Dpt.-Tech. Rep.-823, |Dpt.| Computer Sciences, Purdue University, West Lafayette, IN 47907-2004, USA, 8 |December.| 1988.

[SNS88] J. G. Steiner, C. Neuman and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," in |*Proc.*| *Usenix Winter* |*Conf.*|, pp. 191-201, USENIX Assoc., 1988.

[Sto88] C. Stoll, "Stalking the Wily Hacker," |*Comm. of the ACM*|, vol. 31, no. 5, pp. 484-497, |May| 1988.

[TMR86] A. S. Tanenbaum, S. J. Mullender and R. Renesse, "Using Sparse Capabilities in a Distributed Operating System," in |*Proc.*| *Sixth* |*Int.*| |*Conf.*| *on Distributed Operating Systems*, IEEE, 1986.

[WoK85] P. H. Wood and S. G. Kochan, *UNIX System Security*, Hayden Book Company, 1985.