Sleepy Network-Layer Authentication Service for IPSEC [*]

Shyhtsun F. Wu[1]

Computer Science Department, North Carolina State University, Raleigh, NC 27695-8206. wu@csc.ncsu.edu

Abstract. Network-layer authentication security services are typically pessimistic and static. A conservative IP security gateway checks/verifies the authentication information for every packet it forwards. This implies that, even there is no bad guy in the network, the authentication check is still performed for every packet. In this paper, we examine a sleepy approach, where the gateways normally do not authenticate or verify the packets unless security attacks are detected. We propose a security protocol, *SSGP (Sleepy Security Gateway Protocol)*, residing on top of the *IPSEC (Internet Security Protocol)*. One important feature of SSGP is the collaboration model between network and application layer security mechanisms.

1 Introduction

In the Internet, the *demilitarized zones (DMZ)* separated by firewalls or security gateways[CB94] are static. The system administrators decide the configuration options about the firewalls, which remain unchanged for months or years. For instance, packet filtering firewalls consistently check every incoming packets and reject unauthenticated packets. The checking is done even when there is NO security attack to the networking system.

Traditionally, these gateways are provided only in the transport or application layer. Implementing this type of filtering mechanisms in the *network/IP* layer is beneficial but causes the potential performance problem as mentioned in section 3 of RFC1636 [BCCH94] as well as section 5.3 of RFC1825 [Atk95]. For example, in *swIPe* [IB93], if the security check is done in purely software, then it is measured that we need to pay 100 microseconds per IP packet plus 1 microsecond per byte for MD5 and 10 microseconds per byte for DES. One obvious way to improve the performance is to use some hardware solutions (*e.g.*

[*] This research is supported by Center for Advanced Computing and Communication.

[Ebe92]). However, as pointed out in [Lin94], since the network layer itself generally is implemented in software, it is hard to make effective use of cryptographic hardware.

Generating and verifying the MAC (message authentication code) for every IP packet consume the network resources. The performance of the most commonly used MAC scheme, MD5, can be found in [Tou95]. On the other hand, more and more Internet applications have their own authentication process. For example, Kerberos [NT94], SNMPv2 [GM93], MobileIP's Registration Protocol [Per96], and the next release of Java [Mic, DFW96, WDH+96] have or will have built-in application layer authentication mechanisms. This means that the same IP payload will pass through the MAC process twice, one in the network layer and another in the application layer. The problem is that whether we could do something smart to reduce the security overheads.

In this paper, *SSGP (Sleepy Security Gateway Protocol)* is presented to efficiently offer network-layer authentication services for IPSEC. The key element of SSGP is the *Sleepy Security Gateway (SSG)* which might be in either one of the following two different modes: *sleepy* and *wakeup*. In the following sections, through two different examples, we will show how SSGP works.

2 A Simple Example of SSGP

2.1 Problem

In Figure 1, a system administrator establishes a connection with the management agent across a segment of public Internet. According to the IPSEC architecture, we have two security gateways, SSG_A and SSG_B, and establish a secure channel over the insecure public internet. This scheme is sufficient for outside intruders but not good enough for insider. Therefore, we need an application-layer security mechanism or an intrusion detection module to protect the application against the insider's attacks. Please note that if we only have inside intruders, then the security checks being performed by SSG_A and SSG_B are wasted.

2.2 Answer

Assume that we do have an application-layer mechanism (e.g., SNMPv2) to authenticate the application PDU (e.g., SNMP PDU). Then, even no IP layer security mechanism exists, the application is safe in the sense that it will not accept any unauthenticated packets. This is true for attacks from either the

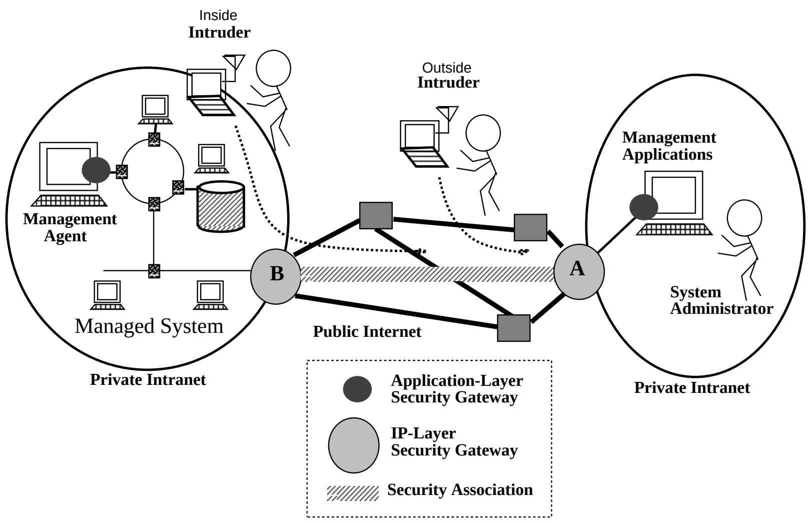

Fig. 1. Application and IP Security

insider or the outsider. However, the application might suffer from heavier incoming traffic load because all the attack packets can now be delivered to the host directly. And, the application needs to spend time to filter out those bad PDUs. For example, without network-layer protection, in [Zor94, Wu95], it is identified that the performance of the SNMPv2 agent drops significantly under denial-of-service attacks.

The application-layer security module detects bad PDUs. For example, SNMPv2 security module dropped packets with wrong key-MD5 authentication and replayed packets. This detection information should be forwarded to the security management module. This management module wakes up sleepy security gateways (SSGs) protecting this application. Based on the *security association* relations defined in [Atk95], these waked-up SSGs might wake up more SSGs. As shown in Figure 2, SSG_A has been waked up by the security management module who received a report from an application. Because a security association exists between SSG_A and SSG_B, SSG_A wakes up SSG_B.

At this point, all the packets from the agent to the administrator are authenticated in the public Internet. The original IP packet from the SNMP agent ($Agent$) to the SNMP management application ($MApp$) looks like:

$$IP(Agent, MApp) = \boxed{Prot_{UDP}, IP_{src}^{Agent}, IP_{dst}^{MApp} : \boxed{IPpayload}}.$$

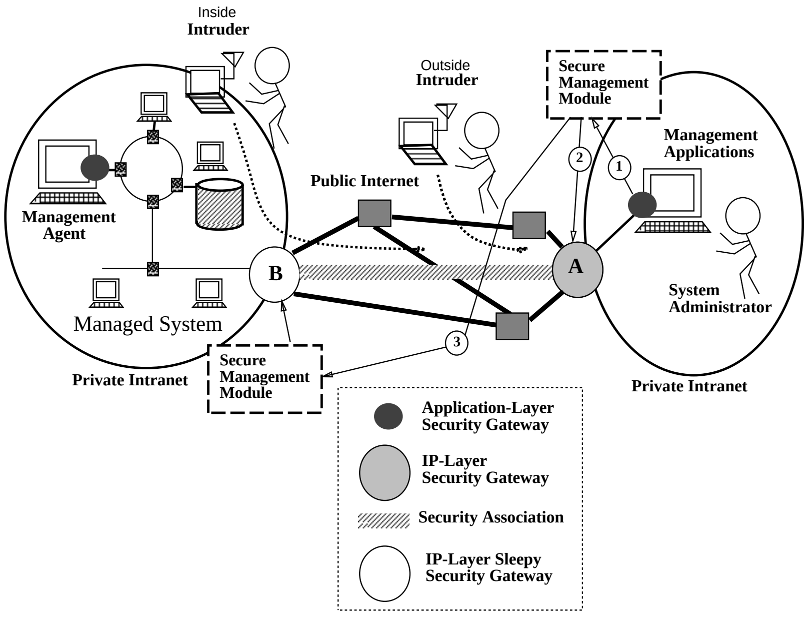

Fig. 2. SSGP: Simple Example

The SSG authenticated packet format, $IP_{encap}(SSG_B, SSG_A)$, is:

$$\boxed{Prot_{SSGP}, IP_{src}^{SSG_B}, IP_{dst}^{SSG_A} : \boxed{\boxed{Auth_{SSG_B}^{SSG_A}} \boxed{IP(Agent, MApp)}}},$$

where the $Auth_{SSG_B}^{SSG_A}$ is

$$\boxed{ID_{SSG_B}, ID_{SSG_A}, Seq\sharp},$$

plus:

$$MD5(K_{AB}, ID_{SSG_B}, ID_{SSG_A}, Seq\sharp, \boxed{IP(Agent, MApp)}, K_{AB}).$$

When this encapsulated IP packet is received by SSG_A, it will be decapsulated by SSG_A and delivered to $MApp$ with the following format:

$$IP_{decap}(Agent, MApp) = \boxed{IP(Agent, MApp)}\boxed{Auth_{SSG_B}^{SSG_A}}.$$

We append the authentication certificate, $Auth_{SSG_B}^{SSG_A}$, at the end of the packet. This information is useful for deciding the attack sources as we will explain later. The reason for appending this certificate at the end of a regular IP packet is to ensure the compatibility between SSGP packets and normal IP packets.

If we only have outside intruders, then, in theory, no more bad packets will get to the protected host $MApp$. If $MApp$ still receives bad packets and SSG_A did NOT find any unauthenticated packets, then we conclude we only have "inside intruders." Otherwise, we are under attacks of both insiders and outsiders simultaneously.

We learn three things from the above results:

1. If no attack exists, then we do not pay the network layer authentication overhead. We always need to pay the application security overhead though.
2. Correlating intrusion information from two different layers, *i.e.,* network and application layers, we will be able to tell the location of the attack sources. This location information is valuable for security management.
3. If we know that the attack sources are purely from insiders, then we know that the network layer authentication does not help at all. In this case, we should put the waked-up SSGs back to sleep again.

3 Another Example of SSGP

3.1 Problem

In Figure 3, we have four SSGs and we assume that the public network is partitioned into four different zones by these SSGs. We assume that there are security associations ($\Leftrightarrow$) between (SSG_A, SSG_B) and (SSG_C, SSG_D). *I.e.,* $SSG_A \Leftrightarrow SSG_C$, $SSG_A \Leftrightarrow SSG_D$, $SSG_B \Leftrightarrow SSG_C$, $SSG_B \Leftrightarrow SSG_D$. However, there is no such relation between SSG_A and SSG_B *i.e.,* $(SSG_A \not\Leftrightarrow SSG_B)$. I.e., the IPSEC traffic from SSG_B to SSG_A must flow through either SSG_C or SSG_D. Previously, we have shown how we can use SSGP to decide whether the attack is from inside or outside. In this example, we would like to decide which zone the attack source resides.

3.2 Answer

At the beginning, all SSGs (A,B, C and D) are in sleepy mode. When the application detects an attack, it will wake up SSG_A first. Then, SSG_A will wake up both SSG_C and SSG_D. The packets authenticated by SSG_C and SSG_D include an authentication path information. Thus, when the application detects an attack, by looking at the authentication certificate appended after the IP packet, we can tell whether this attack packet is from SSG_C or SSG_D. In other words, if

$$IP_{decap}(Agent, MApp) = \boxed{IP(Agent, MApp)}\boxed{Auth^{SSG_A}_{SSG_C}},$$

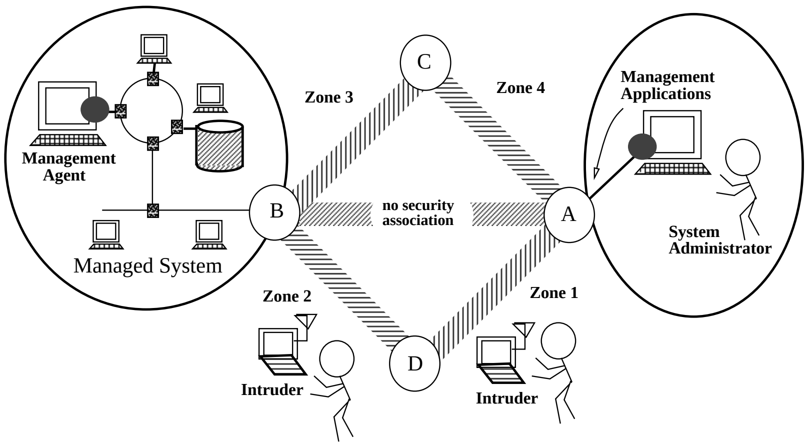

Fig. 3. SSGP: Another Example

then, we know this packet is from SSG_C.

If from now on the application does not observe any attack, while SSG_A detects some unauthenticated IP packets, then we know that the attack source must be either Zone 1 or 4. It depends on the network topology and configuration, and sometimes it may not be possible to further distinguish Zone 1 and 4. In this case, SSG_B will sleep all the time.

Similarly, if the application is still under attack and SSG_A does not detect bad packets, then the attack source must be in either Zone 2, Zone 3 or from an insider. At this point, the attacking packet should be presented to the security management module, and the module can tell whether it is from SSG_C or SSG_D by looking at the appended authentication certificate. If this attack packet is indeed from SSG_C, then SSG_C will wake up SSG_B. Now, an IP packet will pass through two separate IPSEC channels: $SSG_B \Leftrightarrow SSG_C$ and then $SSG_C \Leftrightarrow SSG_A$. The packet from SSG_B to SSG_C is

$$\boxed{Prot_{SSGP}, IP_{src}^{SSG_B}, IP_{dst}^{SSG_C} : \boxed{\boxed{Auth_{SSG_B}^{SSG_C}}\ \boxed{IP(Agent, MApp)}}},$$

where the $Auth_{SSG_B}^{SSG_C}$ is

$$\boxed{ID_{SSG_B}, ID_{SSG_C}, Seq\sharp},$$

plus

$$MD5(K_{BC}, ID_{SSG_B}, ID_{SSG_C}, Seq\sharp, \boxed{IP(Agent, MApp)}, K_{BC}).$$

At SSG_C,

$$IP^{SSG_C}_{decap}(Agent, MApp) = \boxed{IP(Agent, MApp)}\boxed{Auth^{SSG_C}_{SSG_B}}.$$

Then SSG_C will encapsulate the packet again using SSGP and send it to SSG_A. So, $IP_{encap}(SSG_C, SSG_A)$ looks like:

$$\boxed{Prot_{SSGP}, IP^{SSG_C}_{src}, IP^{SSG_A}_{dst} : \boxed{\boxed{Auth^{SSG_A}_{SSG_C}}\boxed{IP^{SSG_C}_{decap}(Agent, MApp)}}},$$

where the $Auth^{SSG_A}_{SSG_C}$ is

$$\boxed{ID_{SSG_C}, ID_{SSG_A}, Seq\sharp},$$

plus,

$$MD5(K_{AC}, ID_{SSG_C}, ID_{SSG_A}, Seq\sharp, \boxed{IP^{SSG_C}_{decap}(Agent, MApp)}, K_{AC}).$$

Finally, at SSG_A, the packet being delivered to $MApp$ is:

$$\begin{aligned} IP^{SSG_A}_{decap}(Agent, MApp) &= \boxed{IP^{SSG_C}_{decap}(Agent, MApp)}\boxed{Auth^{SSG_A}_{SSG_C}} \\ &= \boxed{IP(Agent, MApp)}\boxed{Auth^{SSG_C}_{SSG_B}}\boxed{Auth^{SSG_A}_{SSG_C}}. \end{aligned}$$

Please note that this packet will then have two authentication certificates appended. At this point, it depends on whether we will observe another attack to decide the attack source location.

4 Attack Detection in the Application Layer

SSGP is a lazy security protocol. When no application or intrusion detection system complains about attacks, the SSGs will not perform any authentication check. Thus, when no attack presents, the network is equivalent to an unsecure IP network. At this stage, even *untrusted hosts* can enjoy the network services.

We use SNMPv2 as an example to describe our idea about application-layer attack detection. In Figure 4, the security gateway in the SNMPv2 agent is in the application layer. An SNMPv2 PDU might be rejected for two reasons: authentication failure or freshness constraint.

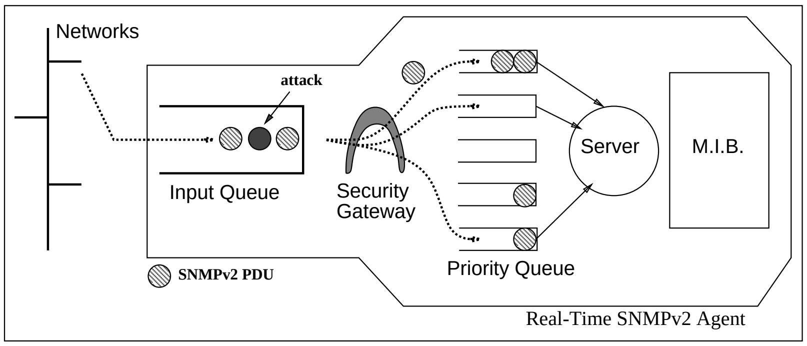

Fig. 4. Real-Time SNMPv2 Agent

4.1 Authentication Failure

If an PDU has an incorrect MD5 signature, this PDU may be an attack, an error in MD5 key configuration, or even a software bug. In any case, we should treat it as an attack and notify the security management module. If this is an error or bug, the SSGP protocol will treat it as an insider attack and, therefore, all *SSG*s should go back to sleep. Then, some network management modules or system administrators should be notified about this problem.

In a wireless network, since the bit error rate is higher, sometimes the link layer checksum mechanism might not correct/detect all the errors in the message. This could also cause incorrect MD5 signatures. Therefore, when accessing SNMP agents across wireless links (*e.g.,* CDPD [CDP95]), we can not conclude immediately the existence of attacks by justing detecting one single bad PDU. It depends on the link error rate to decide how many bad PDUs are enough to notify the management module.

4.2 Freshness Constraint

In SNMPv2, a received PDU might violate the freshness constraint. It could be a replay attack, a duplicated IP packet, or a large network delay. One heuristic is to set a second deadline for the PDU. If the PDU violates the first deadline, it is silently dropped. If it also violates the second deadline, we consider it an attack.

4.3 Attack Detection by Intrusion Detection Systems

Many existing applications do not have any built-in security mechanisms. Therefore, we need to build an intrusion detection system (IDS) [Den87, SRS91] to protect them. For example, in NCSU, we are using SNMPv1 (which has no security by itself) on top of Kerberos. Currently, we are building an IDS for SNMPv1 by monitoring the SNMPv1 traffic in our network.

4.4 Summary

No golden rules exist for the application layer attack detection. We must define the detection policies case by case. For example, to build a detection module for SNMP[WMBL94, WMB93] or Kerberos[NT94, BM90], we need to understand the application layer protocols themselves very well. Furthermore, we need to know what environment we run this application, *e.g.,* ATM or wireless.

5 Security Management Modules (SMM)

Security Management Module (SMM) is a key component in a SSGP system. SMM needs to communicate with both network and application layer security mechanisms. In our current prototype implementation, the SMM will interact with SNMP agents, while the agents collects intrusion information from network and application layers. In this section, the architecture of SMM is presented.

5.1 Intrusion Detection MIB

An *Intrusion Detection MIB (IDMIB)* is an abstraction of detected intrusion with standard management information interfaces (*e.g.,* SNMP or CMIP). An IDMIB specification document should describe what types of information are available in the MIB. For instance, an *intrusion event table (IET)* in the IDMIB collects all the locally detected intrusion events. This table (IET) offers valuable management information for local or even remote SMMs. All information requests from SMMs must be handled by management protocol agents (*e.g.,* SNMP agent).

5.2 Attack Detection

In SSGP, network applications themselves and/or intrusion detection systems are capable of detecting intrusion events. Detected events will be stored in the IET. An IET entry (*i.e.,* a row in the IET) consists of attributes (*i.e.,* columns in the IET) related to a detected intrusion event. These attributes may include the attack PDU with its original IP header, the attack type, and a timestamp.

When the networking system is under serious attacks, IET is updated frequently. And, the information kept in the IET is called *rapidly changing data* [Wu95]. In this case, it is important to maintain the information flows between the IDMIB agents and the SMMs. Sometimes, it is more efficient to have agents notify SMMs about newly detected intrusion events. However, there are also cases where it is better to let the SMMs periodly poll the information out of the IDMIB. In practice, a security management system will perform both polling and event notification.

5.3 Attack Isolation

A local SMM with a local IDMIB sometimes can not handle certain global attacks. For instance, in Figure 2, the SMM behind SSG_A observes attacks, but by itself it can not handle the problem properly. Therefore, SMMs must communicate with each other, exchange intrusion management information, and then a set of collaborating SMMs can decide the correct sources of attacks.

6 Security Consideration

We consider the following security concerns for SSGP:

Security-Sensitive Applications: There are certain important applications that could have very weak security by itself and also it is very hard to build an IDS to protect them. These applications should never be protected by a sleepy security gateway. In other words, the network administrator should configure the SSG such that it will never go to sleep. It is also important to identify what applications are suitable for the SSGP protection and what are not.

Evil Application Detection: If a protected application is evil, then it could complain about attacks while the attacks do not exist. These will wake up SSG gateways unnecessarily. The problem of detecting such evil/faulty applications is another IDS/security management problem. However, in the worst case, the SSGP will just converge to the static IPSEC architecture.

Compromised SSG Gateways: Compromised SSG gateways can wake up other SSG gateways viciously. In fact, without SSGP, if a security gateway in plain IPSEC is compromised, then many serious attacks can happen. To detect this type of problems is extremely important and generally an open problem.

SNMP Security: We use SNMPv2 in our experiments because of its availability today. We expect that a securer version of SNMP will be standardized in the near future.

7 Related Works

Network layer security for IP has been studied in [Bel96, Orm96, CGHK95, WB96, IB93]. Most of these works focus on design and implementation of a network security system similar to IPSEC [Atk95]. The SSGP work presented here went one step further by examining how IPSEC can be used to provide secure and high performance network services.

Security management and intrusion detection system have also been studied extensively (*e.g.,* [Den87, SRS91]). Both logical and statistical approaches have been proposed to handle different types of intrusion. SSGP offers an attractive architecture to integrate IPSEC and the results produced by the IDS community.

8 Conclusion

The IPSEC architecture [Atk95] specified only the security building blocks that we could use for Internet security. It has pointed out certain performance overheads for employing the standardized protocols. However, where and how to put these security building blocks in today's internet is a big open problem. It is very clear that careless placement of these blocks will not secure the network but reduce the available bandwidth. Therefore, in this paper, we propose SSGP as a solution to support secure and high performance authentication services on top of IPSEC. We feel that SSGP is an interesting option for the network administration to protect applications with application-layer security mechanisms.

SSGP defines an architecture about how application and network layer security mechanisms could collaborate under the framework of IPSEC. The collaboration between the security management modules and the SSGP/IPSEC offers the following advantages:

efficiency: SSGP saves unnecessary security checks. The SSG gateways will sleep when either there is no attack or they can not help because it is an insider attack.

isolation: SSGP isolates an outsider attack in its zone by waking up all the SSG gateways around that particular zone.

identification: SSGP identifies the possibility of insider attacks by comparing the security information from both network and application layers.

Currently, we are implementing a prototype version of SSGP/IPSEC and hopefully through this experimental system building process, we will learn more about the strength and weakness of SSGP. Our prototyping experiment is being built on top of Linux PCs with EtherNet and AT&T WaveLan cards. We need to modify the networking module code for participating routers/gateways. We also need to modify the applications so that it will observe/report attacks. Finally, we are using CMU's SNMPv2 package to implement the security management module.

In both SSGP and IPSEC, the insider attack will not be detected until it hits the target host's detection module (*e.g.,* MApp). It will be much nicer, while tracking the insider attacks, if we could perform the application layer detection much earlier. One idea that we are working on is to implement the application detection module as a Java applet. Then, in theory, we could send this applet to the SSG gateway and run it over there. For example, if the attack is from SSG_B in our previous examples, then we could ask SSG_B to load this applet and eliminate the attacks right at that point. The current release of Java does not allow this to happen because of the security concerns of Java itself. We are currently implementing LAVA [WDH+96], a securer version of Java, for this purpose.

Acknowledgments

Matt Blaze encouraged me to look into the problem of performance versus security. Some of the ideas here came out of my Ph.D. thesis which was supervised and shaped by Professor Gail E. Kaiser. Many members in the Secure and Highly Available Networking Group (SHANG) at NCSU have contributed to this work one way or the other. Especially, I want to thank XinYuan Wang, Brian Vetter, Hicong Fu, Tim Sluss and Ilia Baldine.

References

[Atk95] R. Atkinson. Security Architecture for the Internet Protocol. RFC 1825, August 1995.

[BCCH94] R. Braden, D. Clark, S. Crocker, and C. Huitema. Report of TAB Workshop on Security in the Internet Architecture. RFC 1636, June 1994. Network Working Group.

[Bel96] Steven M. Bellovin. Problem Areas for the IP Security Protocols. Draft, March 1996.

[BM90] S.M. Bellovin and M. Merritt. Limitations of the Kerberos Authentication System. *Computer Communication Review*, 20(5):119–132, October 1990.

[CB94] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security.* Addison Wesley, 1994.

[CDP95] Cellular Digital Packet Data System Specification. CDPD Forum, Release 1.1, January 1995.

[CGHK95] Pau-Chen Cheng, Juan A. Garay, Amir Herzberg, and Hugo Krawczyk. Design and Implementation of Modular Key Management Protocol and IP Secure Tunnel on AIX. In *1995 IEEE Symposium on Research in Security and Privacy*, May 1995.

[Den87] Dorothy Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, February 1987.

[DFW96] D. Dean, E.W. Felten, and D.S. Wallach. Java Security: From HotJava to Netscape and Beyond. In *1996 IEEE Symposium on Security and Privacy*, pages 190–200, May 1996.

[Ebe92] Hans Eberle. A High-Speed DES Implementation for Network Applications. Technical Report 90, DEC SRC, Palo Alto, CA, September 1992.

[GM93] J. Galvin and K. McCloghrie. Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1446, May 1993. Network Working Group.

[IB93] John Ioannidis and Matt Blaze. The Architecture and Implementation of Network-Layer Security Under Unix. In *4th Usenix Security Symposium*, October 1993.

[Lin94] Mark H. Linehan. Comparison of Network-Level Security Protocols. Technical Report White Paper, IBM Research Division, June 1994.

[Mic] Sun Microsystems. HotJava(tm): The Security Story. Java home page.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, pages 33–38, September 1994.

[Orm96] Hilarie Orman. Evolving an Implementation of a Network Level Security. Technical Report TR-95-15, CS Department, University of Arizona, Tucson, AZ, January 1996.

[Per96] Charles Perkins. IP Mobility Support, draft 15. Internet Draft, IETF, February 1996. Mobile IP Working Group.

[SRS91] et. al. Steven R. Snapp. DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and an Early Prototype. In *14th National Computer Security Conference*, pages 167–176, October 1991.

[Tou95] Joseph Touch. Performance Analysis of MD5. In *SIGCOMM: Communications Architectures, Protocols and Applications*, pages 77–86, Cambridge, MA, September 1995.

[WB96] David A. Wagner and Steven M. Bellovin. A Bumper in the Stack Encryptor for MS-DOS Systems. In *IEEE Symposium on Network and Distributed Systems Security*, 1996.

[WDH$^+$96] S. F. Wu, M.S. Davis, J. Hansoty, J. Yuill, J. Webster, and X. Hu. LAVA: Secure Delegation for Mobile Applets. Technical Report TR-96-05, Computer Science Department, North Carolina State University, June 1996. submitted.

[WMB93] Shyhtsun F. Wu, Subrata Mazumdar, and Stephen Brady. EMOSY: An SNMP Protocol Object Generator for the PIMIB. In *IEEE First International Workshop on System Management*, Los Angeles, CA, April 1993.

[WMBL94] Shyhtsun F. Wu, Subrata Mazumdar, Stephen Brady, and David Levine. On Implementing a Protocol Independent MIB. In *Network Management and Control, volume 2*, pages 309–329. Plenum Press, New York, 1994.

[Wu95] Shyhtsun F. Wu. *ϵ-Consistent Real-Time Monitoring for Rapidly Changing Data*. PhD thesis, Columbia University, Computer Science Department, New York, NY, July 1995.

[Zor94] Vasilios Zorkadis. Security versus Performance Requirements in Data Communication Systems. In *Computer Security - ESORICS 94*, pages 19–30, Brighton, UK, November 1994.

This article was processed using the LaTeX macro package with LLNCS style