Implementing a Secure rlogin Environment: A Case Study of Using a Secure Network Layer Protocol

Gene H. Kim, Hilarie Orman, and Sean O'Malley
University of Arizona, Tucson, AZ

Implementing a Secure rlogin Environment: A Case Study of Using a Secure Network Layer Protocol

Gene H. Kim Hilarie Orman Sean O'Malley

Department of Computer Science
University of Arizona
Tucson, AZ 85721

Abstract

This paper describes our experiences building a secure *rlogin* environment. With minimal changes to the *rlogin* server and the use of a secure network layer protocol, we remove the vulnerability of hostname-based authentication and IP source address spoofing. We investigate how applications such as *rlogin* interact with this new layer, and propose extensions to the *rlogin* server that can utilize these services. We believe *rlogin* presents a situation where the application layer seems the most appropriate location for enforcing security policy, instead of in a lower layer.

Our layered approach to *rlogin* security achieves functionality similar to the Kerberos *klogin* client and the encrypted *telnet* packages, without their complexity or loss of generality. Implementing the application layer *rlogin* server extensions required fewer than ninety lines of code. Even if our *rlogin* application layer extensions are omitted, *rlogin* connections still benefit from secure network layer services.

1 Introduction

The *rlogin* program provides a security feature and at the same time introduces vulnerabilities with respect to user authentication on UNIX systems. The security feature uses the notion of *trusted hosts* to avoid sending passwords over a network, where they are vulnerable to exposure. However, flaws in the mechanism can be exploited to allow malicious users to gain entry to machines. These security vulnerabilities in the TCP/IP protocols and the Berkeley "r-command" programs have been discussed at length in the literature [3, 20, 27]. Nonetheless, programs such as as *rlogin* remain in widespread use. The consequences of not eliminating these security vulnerabilities can be seen in the recent CERT advisory [6] pertaining to widespread abuses of Unix systems via IP host spoofing and TCP connection forging.

In this paper, we address an attack whereby a malicious host masquerades as a trusted host to bypass the *rlogin* password security mechanism and gain access to the system. The attacker can either forge DNS name and address resolution replies [3] or forge the source address in the IP packet header. The latter attack is not a weakness due to hostname based authentication, but is a more general problem of remote host authentication.

In response to these attack scenarios, we describe our experiences building a secure *rlogin* environment using a prototype secure network layer protocol. This layer provides services for remote host authentication, message integrity, and optionally, message privacy (i.e., packet encryption). Using these services, we have built and demonstrated an *rlogin* server that is protected from attacks and protects passwords from exposure under more circumstances.

Our approach to *rlogin* security is completely modular. By using the *x*-kernel [1] as our design framework, the network security layer can provide authentication and privacy services without changing application layer programs. Other solutions, such as Kerberos[15] and the encrypted *telnet* programs discussed in [23, 26], require modifying existing server and client programs. Our modifications to *rlogin* allow it to establish a more flexible security policy than it did previously.

We attribute the simplicity of our implementation and the concise interface between the application and secure network layer to our software methodology. We believe that modular protocols can assist in providing network security enhancements, and that small, well-defined module interfaces are sufficient even at the application level.

In the following sections, we describe the security vulnerabilities in widely used UNIX network protocols and programs that can be exploited by malicious users to gain unauthorized access. Next, we describe our design of the *rlogin* server and its extensions, and we present our experiments with an *rlogin* client that establishes a TCP connection with forged IP and Ethernet source addresses to spoof remote *rlogin* servers.

We then describe other solutions that have been proposed to solve the problem of securing remote login sessions. We conclude our paper by comparing our design choices with these other works, addressing the issues of placement of policy manager and the interface between the application and the network layer.

2 Problem definition

Many risks to system security stem from design flaws in network protocols, servers, and programs originally written a decade ago. In the following sections, we describe vulnerabilities that are related to the use of *rlogin*.

2.1 Password Vulnerability

When an *rlogin* server establishes a connection with a non-trusted host, it prompts the user for a password. As the user types the password, it is transmitted over the network. In the past two years, this vulnerability has extensively exploited [5] to capture passwords *en masse*.

2.2 Host Naming Vulnerabilities

In the *rlogin* server, trusted hosts are referenced by their names. Resolving these hostnames into IP addresses (and vice versa) is accomplished by the Domain Naming System [19], which refers to both the hierarchical, distributed database and the query-response protocol for accessing it. At the current time, there is no assurance that the information in a DNS response is valid. Consequently, any security mechanisms depending upon mapping names to addresses are ill-founded. This is documented in the literature [3, 27], and has been exploited in system attacks[11].

2.3 TCP/IP Vulnerabilities

TCP is used as transport service for *rlogin*. TCP and IP vulnerabilities to source address spoofing and connection forging are discussed in [3, 20]. Until recently, these vulnerabilities may have seemed too obscure to warrant concern. However, in January 1995, a CERT computer security advisory [6] described the availability of tools that automate the process of IP host spoofing and TCP connection forging — malicious users now can use these techniques without detailed knowledge of their operation. How this tool can be used to gain privileged access on UNIX machines running *rlogind* is described in the next section.

2.4 Rlogin: Password Protection and Vulnerabilities

The 4.2BSD operating system [16] introduced a suite of programs (sometimes called the "r-commands") that allowed the convenient access of remote resources on network-accessible machines. These programs, which include *rcp*, *rsh*, and *rlogin*, establish remote sessions with an associated user identifier that is used for access control decisions; typically the identifier is the same on both machines.

The conventional *login* program, executed when logging onto a machine on its console or via *telnet*, requires users to type their passwords. If transmitted over a network, the password is subject to exposure as described previously. To prevent this, the *rlogin* program allows host authentication to substitute for a password. If a user is already authenticated as a user on a trusted host, then this authentication can be adopted without password verification on the target host. This transitive trust relationship is a discretionary policy determined by the site administrator in the */etc/hosts.equiv* file and by individual users via their *.rhosts* files.

Although the avoidance of password exposure prevents eavesdroppers from learning passwords, the method of authenticating the originating host is so weak that it has become a major vulnerability. There are two aspects to the vulnerability. First and most basic, no mechanism (cryptographic or otherwise) validates whether a network packet originated from the host indicated in the packet as the "source." Second, trusted hosts are referenced by their host names, and the mechanism for translating a name to an Internet address is not secure. As a result, any host on the Internet can send packets masquerading as another host, whether trusted or not.

3 Design and implementation

The work described here addresses the vulnerabilities described above. Strong cryptography is used for trusted host authentication, and an alternative form of password protection is provided to encrypt all packets in a key that is negotiated by the two hosts.

There are three parts to the design: the modular protocol framework, the secure network layer implemented within that framework, and the *rlogin* server, which applies the discretionary policy based on information provided to it by the secure network layer.

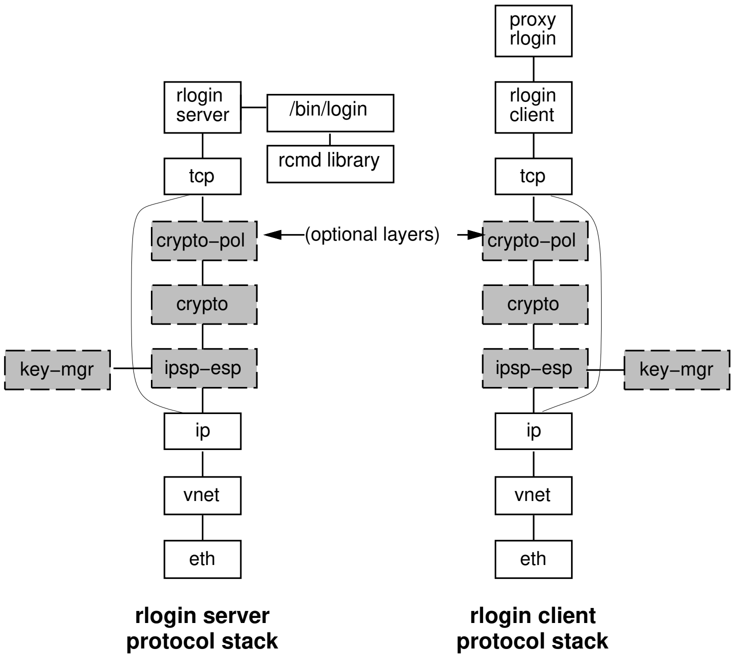

Figure 1: Protocol stacks for *rlogin* server and client

3.1 Modular protocols

Our design and implementation framework used the object-oriented *x*-kernel networking architecture [21]. The *x*-kernel facilitates the development of layered network protocols — protocols are small and modular, with fixed, well-defined operations between the layers. The advantages of such an approach are more fully discussed in [22].

We implemented a substantial subset of the *rlogin* server and client functionality in top layer protocols in the *x*-kernel networking architecture. Implementing the *rlogin* server and client as *x*-kernel protocols allowed us to determine to what extent these particular application layer programs interact with the secure network layer.

Beneath the *rlogin* protocols is the standard TCP/IP protocol stack, with the optional inclusion of our prototype secure network layer protocol. A diagram of the major protocol modules is shown in Figure 1. Note that each box in the diagram represents an individual software module presenting *x*-kernel interface functions for opening and closing connections, sending and receiving messages, and answering queries about connections attributes (e.g. maximum packet size, type of security, etc.). Note also the pathway that bypasses the network security layer; this allows backwards compatibility for interoperating with hosts that do not implement the network security layer.

3.2 Secure network layer

Our initial version of network layer security provides cryptographic security enhancements for the data portion of IP version 4 packets. The security of the enhancements depends on a pairwise shared key between hosts; the keys can either be manually pre-distributed or dynamically negotiated.

The algorithms accepted between two hosts are determined by the site configuration module. For instance, a site may require that MD5 [25] authentication be used for all packets to and from a particular remote host. Only those packets that are acceptable according to the site policy and pass the required cryptographic checks are allowed to proceed up the protocol stack (i.e., to higher level protocols and applications).

Algorithms for all combinations of sender authentication, message integrity, and message privacy are available. In practice, sender authentication and message integrity are provide by using a keyed hash function such as MD5. Although the option of message privacy alone is provided by using DES in CBC mode [9], this combination is not generally recommended [4]; privacy plus authentication and integrity (e.g. DES-CBC over unkeyed MD5) is a better option because it protects packets against having information appended to them without detection. Nonetheless, for the purpose of protecting passwords, the DES-CBC option is viable, and

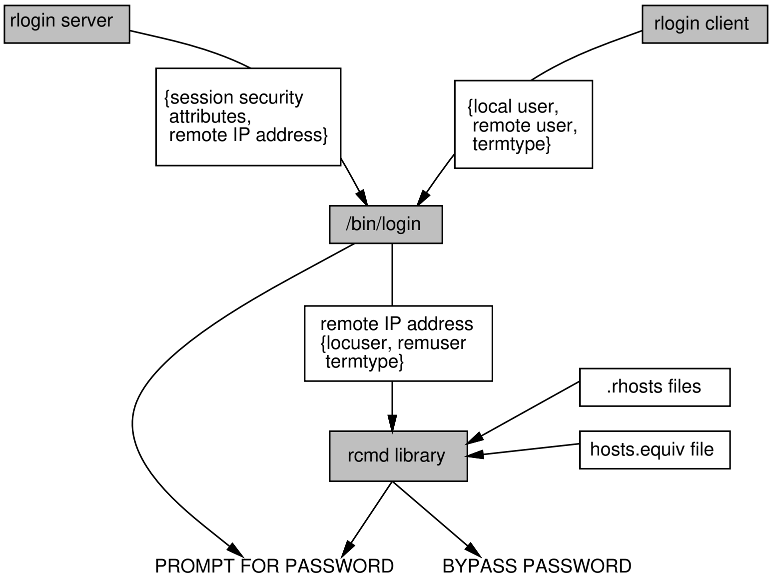

Figure 2: Flow graph for *rlogin* server authentication

rlogin makes use of it.

It should be noted that when a message integrity service and sender authentication service are provided by the secure network layer, via keyed MD5 for example, then all packets on that TCP connection are protected, not just the initial packets for user authentication. This implicitly protects against the attacks on TCP connections via IP address spoofing.

The *x*-kernel structure allows applications to query lower protocols for attributes of a connection. For our secure network layer, there are two relevant attributes: sender authentication and data privacy. The *rlogin* protocol makes use of these attributes in applying its discretionary access policy.

Currently, host-host key management is done with Diffie-Hellman key exchange [10] and validated using RSA signatures [24]. For this prototype implementation, RSA public keys are manually distributed.

Most aspects of the network layer security policies are determined by information in configuration files; at runtime, the configuration information is made available by management modules using the uniform protocol interface of the *x*-kernel .

3.3 Rlogin server

To take full advantage of the security services provided by the network layer, we modified the *rlogin* server in three ways: it queries the network layer for the security attributes of the connection, it uses this information to implement a richer authentication and password protection scheme, and only Internet addresses (not names) are allowed for authentication decisions.

3.3.1 Unmodified rlogin server operation

In the Berkeley implementation of the *rlogin* server, functionality is divided between the *rlogind* daemon, *login*, and the *rcmd* library residing in *libc*. How the functions are distributed is summarized in Figure 3. The *rlogind* daemon waits on a well-known port for incoming connections. Once a TCP connection is established, the daemon *forks* and executes *login* , passing the remote hostname as an argument. As part of the connection establishment handshake, *login* will receive from the remote *rlogin* client the tuple {*local user, remote user, termtype*}.

To determine whether the remote host is a trusted host, *login* passes the tuple and the remote hostname to the *ruserok()* function residing in the *libc* library. In other words, if the hostname is a trusted host listed in the system *hosts.equiv* file, or if the {*hostname, local user*} pair is in the user's *.rhosts* file, the user is logged in without any password authentication. Otherwise, *login* prompts for a password. (A flow graph of *rlogin* authentication information is shown in Figure 2.)

As a measure to protect against bogus DNS responses, the *rlogin* server attempts to validate the resolved address by checking the inverse mapping. First,

Component	Functions
rlogind	connection establishment
login	calls *rcmd()* to decide whether to prompt for password
rcmd library call	examines .rhosts files to discover trust domains

Figure 3: Breakdown of Berkeley *rlogin* server components

```
quercus.CS.Arizona.edu    gkim
# quercus' IP address
192.12.69.73              root    AUTH|PRIVACY
```

Figure 4: A new *rhosts* file

the server gets the remote host address from the IP packet header. Using this address, a host name is resolved by calling *gethostbyaddr()*. The inverse address mapping is then resolved using *gethostbyname()*. If the two addresses do not match, or if the resolved host name does not match any trusted host name in the *.rhosts* file, the connection is terminated.

Our implementation of the *rlogin* server is a functional subset of Berkeley implementation — a UNIX *rlogin* client can connect with our *x*-kernel version of the *rlogin* server. Most feature omissions pertain to terminal I/O services typically provided through the STREAMS interface. Our implementation does, however, mimic the mechanisms for connection establishment and refusal, password bypass policy, and the I/O stream to the local user shell. Server extensions were added to address security vulnerabilities.

3.3.2 Removing hostname references

Authenticating a remote host by its hostname introduces the risk of DNS spoofing. To obviate this risk, we modified the interface to *login*, replacing the remote hostname argument with the remote host address. There were few changes necessary to implement this, but they required changing the *ruserok()* library call to allow the parsing of IP addresses in the *.rhosts* and *host.equiv* files. Note that this change alone does not eliminate the vulnerability of IP source address spoofing.

3.3.3 Rhosts policy extensions

A new field in the *.rhosts* file specifies the security attributes that must be present in an incoming connection to bypass the password mechanism. At present, these attributes are **`AUTH`** and **`PRIVACY`**, specifying remote host authentication and connection encryption services respectively. An example of a new *rhosts* file is shown in Figure 4. The security services being provided by the secure network layer for the connection are passed by the *rlogin* server as a additional arguments to *login*. For example, *login* may be invoked with "/bin/login -r remotehost -auth -priv", which denotes a remote *rlogin* session from *remotehost* with an authenticated and encrypted connection. If the specified security services do not meet the requirements specified in the *rhosts* file, the connection is terminated. These extensions allow users to create separate discretionary access policies for trusted, unauthenticated, and/or untrusted hosts.

Implementing the changes to the application level *rlogin* server required fewer than ninety lines of code. In the Berkeley implementation, these changes would have modified source files pertaining to the *rlogind* daemon, */bin/login*, and the *rcmd* service in the **`libc`** library.

3.4 Allowed policies

To determine the security attributes of an incoming connection, the *rlogin* server uses control operations to query for the presence of sender authentication and/or privacy. The network security layer supplies replies to these queries; the security of the authentication depends on the key distribution mechanism for setting up pairwise keys that hosts can use for identifying each other. The privacy attribute is independent of the authentication.

Our *rlogin* server can be configured to reject all incoming connections that cannot be authenticated (i.e., not using the secure network layer protocol), and also reject connections based on the security requirements per-user/per-host listed in the user *rhosts* files. Between these two mechanisms, our *rlogin* server can be configured with the same granularity as Kerberos *klogin* or encrypted *telnet* programs.

In Figure 5, we enumerate the *rlogin* server actions as a function of authenticated and unauthenticated clients, and the security flags specified in the user's *rhosts* file.

3.5 Demonstration

To demonstrate the effectiveness of an *rlogin* server running on top of a secure network layer, we

Authenticated remote host	*rhosts* specifications			Description
	trusted	AUTH	PRIVACY	
-	-	-	-	Prompt for password *(Equivalent to conventional Berkeley rlogin .)*
-	yes	-	yes	Prompt for password *Encryption protects against password eavesdropping*
-	yes	-	-	Prompt for password *Bad site policy — suitable for non-essential machines offering low-integrity public services. Password is never sent in the clear.*
-	yes	yes	yes	Reject connection
-	yes	yes	-	Reject connection
yes	-	-	yes	Prompt for password *Encryption protects against password eavesdropping*
yes	yes	-	yes	Bypass password authentication
yes	yes	yes	yes	Bypass password authentication

Figure 5: Possible connection configurations

ran a special *rlogin* client that forged a trusted host's IP and Ethernet source address. This client would connect with an *rlogin* server, and log into the system without supplying a password. A conventional TCP connection is maintained by putting the host's network interface into promiscuous mode, and having the Ethernet and IP layers passing up packets intended for the host being impersonated. The address spoofing modifications to the *rlogin* client required fewer than fifty lines of changes.[1]

We were consistently able to circumvent password authentication in the Berkeley *rlogin* server. As expected, we were unable to be authenticated by the secure network layer residing under our *x*-kernel *rlogin* server. The server accepts an argument to enter one of two modes: a paranoid mode that allowed only authenticated connections, and another that allowed any connection. By specifying minimum security requirements in the third field of the *rhosts* file, users can control with finer granularity the level of security for remote hosts and users.

Our experiments were run with the *x*-kernel, version 3.2 under Mach v3.82 and SunOS v4.1.1. These ran on DECstation 5000/25s and Intel 486 machines for Mach, and Sun SparcStation 2s otherwise.

4 Other work

Some of the other solutions that have been proposed to reduce the risk of system penetration via the network use specialized hardware, additional network layers, and new applications.

4.1 Router-based solutions

CERT currently prescribes the use of filtering routers [23, 8] to reduce a site's vulnerability to IP source address spoofing. Filtering routers and firewalls can be configured to prevent incoming IP packets with internal source addresses from entering the network. However, because these mechanisms operate by interposing themselves between the source and destination hosts, machines within a trusted network can still spoof each other. Because firewalls must be in the routing path, vulnerabilities still exist.

The more sophisticated firewalls now becoming available can also provide point to point IP encryption services [2]. However, the scope of protection afforded is similar to the case of filtering routers — packets are still sent in the clear by the originating host, presenting a window of vulnerability to eavesdropping.

These remaining issues in router based solutions have helped motivate another approach to providing network security. Many network practitioners now believe that providing a mechanism for *end-to-end security* represents the best solution for eliminating the vulnerabilities described in the previous section, typically provided by a secure network layer.

[1] The small number of lines changed does not reflect the amount of effort required to get the hardware and Mach device drivers working correctly!

4.2 Secure network layer solutions

Network layer security is a generic approach that can provide security enhancements for many applications, and validation of this claim was a factor in our choice of *rlogin* as a guinea pig protocol for our prototype network layer security protocol.

In 1992, the Internet Protocol Security Working Group was formed to "develop mechanisms to protect client protocols of IP" at the network layer [12]. Proposed network layer services include message privacy, message integrity, source machine and network authentication, access control, reflection protection, security labels, padding, and methods of avoiding traffic analysis.

[17] presents a survey of protocols that have been submitted to date. Although no proposal has been chosen for adoption at this time, prototypes have been built and demonstrated. Among them are swIPe[14] and IPST[7], as well as our secure network layer protocol described in this paper.

The secure network layer service needed to eliminate *rlogin* vulnerabilities to IP spoofing is host authentication, and optionally, message privacy (i.e., packet encryption). In our work, message privacy is used as a way of protecting transmitted passwords.

4.3 The Kerberos approach

Kerberos [15] takes a different approach to authentication, using a trusted third-party to provide all user and host authentication services — authentication tickets are granted on a per-user, per-host basis for a given service. Hosts and users are authenticated by their knowledge of a host-specific or user-specific secret, respectively.

The primary disadvantage of Kerberos is that all clients requiring authentication services must be modified to use Kerberos services. For example, the Kerberos package includes replacements for some of the r-commands, but it requires the installation of a centralized ticket server. Because of this, and the associated problems of scaling this approach to the entire Internet, Kerberos may not be a generally feasible solution.

4.4 Encrypted telnet packages

Telnet modifications that incorporate encryption and authentication services are described in [23, 26]. This approach incorporates all aspects of network security at the application layer. Instead of delegating these functions to a package such as Kerberos, or delegating them to a network layer protocol, these packages reimplement them on a per-application basis. As a result, considerable functional redundancy may exist.

4.5 Secure DNS

The IETF has also formed a working group to address the security vulnerabilities in the Domain Name Service hostname resolution protocol [19]. At the time of this writing, [13] has been submitted to the DNS Security Working Group. However, to reduce the set of dependencies, we do not utilize any of these services. Use of secure DNS is not incompatible with our approach to network layer security; indeed, we plan to use DNS extensions for accessing the public keys that are necessary for assuring authentication in our key exchange protocol.

4.6 The GSS Application Layer Interface

For purposes of comparing application interfaces, we briefly describe the Generic Security Service Application Program Interface described in [18]. It is intended to support in a generic manner cryptographically oriented security services, such as authentication, integrity, non-repudiation, and privacy. Programs using GSS-API benefit from source-level portability of applications, and independence from the underlying security mechanisms.

The scope of GSS-API is quite large, providing thirteen major control operations, which return one of twenty-four defined return values. The operations manage credentials and security contexts, operate on generic data objects, and provide ancillary support.

5 Discussion and analysis

Our changes to the *rlogin* application layer server removed hostname based authentication and added facilities to describe minimal security requirements on a per-host basis. It was not strictly necessary to change the application layer programs; the reasons for doing so deserve explanation.

Network layers such as swIPe do not require application layer changes. Instead, they encapsulate policy within the network security layer, refusing connections based on the remote host address. A similar mechanism could be provided by a security manager protocol residing between the network and application layer, only allowing connections meeting certain parameters.

However, we assert that all remote login connections are not equal, even if they originate from the same remote host. Consider the case when one *rlogin* session prompts for a password and another automatically accepts the user without prompting for a password. In the first case, encryption must be provided to protect the transmission of the password. In the second case, authentication services must be provided, but there may be no clear need for encryption. Clearly, the policy decision cannot be made at the network security layer for lack of information.

In our case, implementing policy at the application layer does not require much communication with the underlying protocol layers. The extent of interaction between the *rlogin* server and the underlying protocols is very small: the two control operations for determining source address authentication and packet privacy. The first operation queries whether an underlying layer is providing remote host authentication services. The operation is propagated down the protocol stack until a protocol services the control request. If the operation is propagated down the entire protocol stack without being serviced, a failure message is automatically generated.) The second operation determines if encryption is being applied to the packets on the connection.

When compared to large number of interface calls defined in the GSS-API, our application interface seems Spartan. However, we believe it is no less versatile. The IETF specification for IP security requires user-oriented keying, whereby hosts would guarantee that different users have different keys for their connections; our prototype is being extended to handle this, and two new interface operations will be provided to applications for getting and setting the key identifiers for their connections.

As described, our *rlogin* implementation allows specifications of policies based on remote hosts and remote users with the same granularity as Kerberos. This policy does not require the strong user authentication and centralized key servers that underlie Kerberos.

We attribute the simplicity of our *rlogin* server implementation and the concise interface between the application and secure network layer to our software methodology. It demonstrates that a small, fixed interface is sufficient even at the application level, and that modular protocols assist in providing network security enhancements.

6 Conclusions

In this paper, we have described our approach to solving *rlogin* security vulnerabilities via a secure network layer, avoiding extensive modifications to application level programs. We describe a set of *rlogin* server extensions (e.g., adding a third field to user *rhosts* files) that allow system administrators to specify policy at a fine-grain level, equivalent to the per-user/per-host model of Kerberos, while retaining simplicity. This is an example of a useful policy that cannot be enforced at the network security layer.

7 Acknowledgements

We thank Andrey Yeatts for providing his Mach device driver and hardware expertise. Without his time, we would still be trying to get our demonstration working. We also thank Rich Schroeppel for reviewing this paper.

References

[1] Mark B. Abbott and Larry L. Peterson. A language-based approach to protocol implementation. *IEEE/ACM Transactions on Networking*, 1(1):4–19, February 1993.

[2] Frederick M. Avolio and Marcus J. Ranum. A network perimeter with secure external access. Technical report, Trusted Information Systems, Jan 1994.

[3] Steve Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communications Review*, Vol. 19(No. 2), April 1989.

[4] Steve Bellovin. Hostpair weaknesses, ipsec discussion. Technical report, ATT Bell Laboratories, 1995.

[5] CERT Coordination Center. *CA-94:01: Ongoing Network Monitoring Attacks*, 1994.

[6] CERT Coordination Center. *CA-95:01: IP Spoofing Attacks and Highjacked Terminal Connections*, 1995.

[7] Pau-Chen Cheng. Design and implementation of modular key management protocol and ip secure tunnel on aix. In *Proceedings of the Fifth Usenix Unix Security Symposium*, 1995.

[8] Bill Cheswick and Steve Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.

[9] Data encryption standard. National Bureau of Standards FIPS, 1977.

[10] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information*, IT-22, Nov 1976.

[11] Mark W. Eichin and Jon A. Rochlis. With microscope and tweezers: An analysis of the internet virus of november 1988. In *Proceedings of the 1989 IEEE Symposium in Research on Security and Privacy*, 1989.

[12] Internet Engineering Task Force. Internet protocol security protocol working group charter. *Internet Activities Board*, 1992.

[13] Donald E. Eastlake III. Domain name system protocol security extensions. Technical Report IETF Working Draft draft-ietf-dnssec-secext-03.txt, DNS Security Working Group, Jan 1995.

[14] John Ioannidis and Matt Blaze. The architecture and implementation of network-layer security u nder unix. In *Proceedings of the Fourth Usenix Unix Security Symposium*, pages 29–39, October 1993.

[15] J. Kohl and Clifford Neuman. The kerberos network authentication service (V5). Request for Comments (Proposed Standard) RFC 1510, Internet Engineering Task Force, September 1993.

[16] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, 1989.

[17] Mark H. Linehan. Comparison of network-level security protocols. Technical report, IBM T. J. Watson Research Center, June 1994.

[18] J. Linn. RFC 1508: Generic security service application program interface, version 2. *Internet Activities Board*, November 1994.

[19] P. Mockapetris. RFC 1034: Domain names — concepts and facilities. Technical report, Internet Activities Board, November 1987.

[20] Robert T. Morris. A weakness in the 4.2bsd unix tcp/ip software. Technical report, ATT Bell Laboratories, 1985.

[21] Sean W. O'Malley and Larry L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10(2):110–143, May 1992.

[22] H. Orman, S. O'Malley, R. Schroeppel, and D. Schwartz. Paving the road to network security, or the value of small cobblestones. In *Proceedings of the 1994 Internet Society Symposium on Network and Distributed System Security*, February 1994.

[23] Marcus J. Ranum. Thinking about firewalls. In *Proceedings of Second International Conference on Systems and Network Security and Management (SANS-II)*, Apr 1994.

[24] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, Vol. 21(No. 2), 1978.

[25] R. L. Rivest. RFC 1321: The md5 message-digest algorithm. Technical report, Internet Activities Board, April 1992.

[26] David R. Safford, Douglas Lee Schales, and David K. Hess. The TAMU security package: An ongoing response to internet intruders in an academic environment. pages 91–118, Berkeley, CA, 1993. USENIX Association.

[27] Christoph L. Schuba and Eugene H. Spafford. Countering abuse of name-based authentication. Technical Report CSD–TR–94–029, COAST Laboratory, Purdue University, apr 1994.