A Toolkit and Methods for Internet Firewalls

Marcus J. Ranum

Frederick M. Avolio

Trusted Information Systems, Inc.

Abstract

As the number of businesses and government agencies connecting to the Internet continues to increase, the demand for Internet firewalls — points of security guarding a private network from intrusion — has created a demand for reliable tools from which to build them. We present the TIS Internet Firewall Toolkit, which consists of software modules and configuration guidelines developed in the course of a broader ARPA-sponsored project. Components of the toolkit, while designed to work together, can be used in isolation or can be combined with other firewall components. The Firewall Toolkit software runs on UNIX® systems using TCP/IP with the Berkeley socket interface. We describe the Firewall Toolkit and the reasoning behind some of its design decisions, discuss some of the ways in which it may be configured, and conclude with some observations as to how it has served in practice.

Overview

Computer networks by their very nature are designed to allow the flow of information. Network technology is such that, today, you can sit at a workstation in Maryland, and have a process connected to a system in London, with files mounted from a system in California, and be able to do your work just as if all of the systems were in the same room as your computer. Impeding the free flow of data is contrary to the basic functionality of the network, but the free flow of information is contrary to the rules by which companies and governments need to conduct business. Proprietary information and sensitive data must be kept insulated from unauthorized access yet security must have a minimal impact on the overall useability of the network.

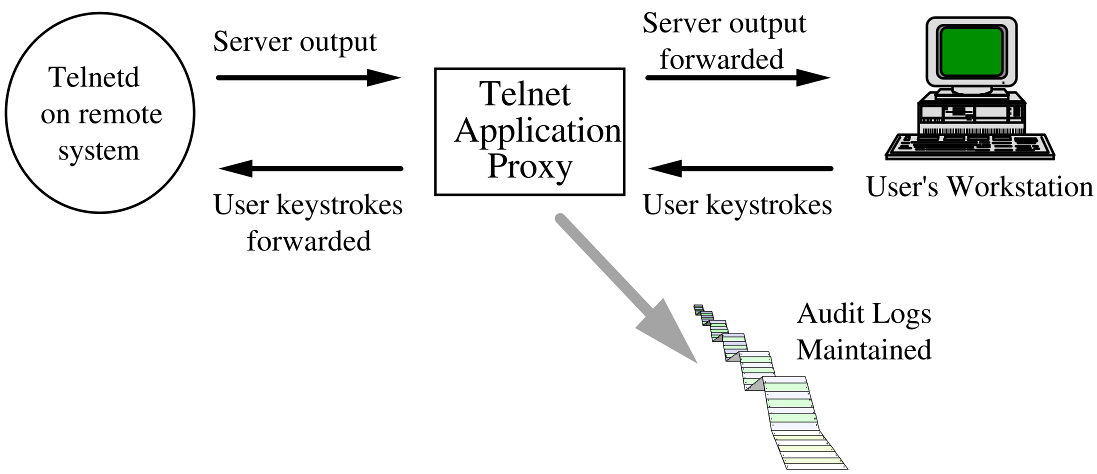
The purpose of an Internet firewall is to provide a point of defense and a controlled and audited access to services, both from within and without an organization´s private network. This requires a mechanism for selectively permitting or blocking traffic between the Internet and the network being protected[1]. Routers can control traffic at an IP level, by selectively permitting or denying traffic based on source/destination address or port. Hosts can control traffic at an application level, forcing traffic to move out of the protocol layer for more detailed examination. To implement a firewall that relies on routing and screening, one must permit at least a degree of direct IP-level traffic between the Internet and the protected network. Application level firewalls do not have this requirement, but are less flexible since they require development of specialized application forwarders known as "proxies." This design decision sets the general stance of the firewall, favoring either a higher degree of service or a higher degree of isolation. [1]

A proxy for a network protocol is an application that runs on a firewall host and connects specific service requests across the firewall, acting as a gateway. Figure 1 represents a minimal TELNET service proxy, in which the proxy forwards user´s keystrokes to a remote system, and maintains audit records of connections. Proxies can give the illusion to the software on both sides of a direct point-to-point connection. Since many proxies interpret the protocol that they manage, additional access control and audit may be performed as desired. As an example, the FTP proxy can block FTP export of files while permitting import of files, representing a granularity of control that router-based firewalls cannot presently achieve. Router-based firewalls can provide higher throughput, since they operate at a

[1] Or, in general, between any two networks where one needs to be protected from the other.

protocol level, rather than an application level, but practical experience running firewalls on modern RISC processors shows that with a T-1 connection, the bottleneck tends to remain the T-1 link rather than the firewall itself.

Figure 1: An Application Proxy



Proxies exist for a wide variety of services, such as X, FTP, TELNET, etc. Perhaps the most significant security benefit of employing proxies is that they provide a convenient opportunity to require authentication. For example, when connecting into a protected network from the Internet, one must typically first connect to the proxy, authenticate to it, and then complete a connection to a host within the protected network. The proxy protects the firewall host itself, by eliminating the need for the user to log into the firewall itself, and it protects the network by permitting only authenticated users to gain access from the outside. While hosts on the private network may still be rife with security holes, restricting the incoming traffic to authenticated users only is a good step in the right direction.

Other services, such as Internet (SMTP) mail and USENET news, act as store-and-forwarders already, and fit in with the proxy approach to firewalls. These service daemons sometimes run with system privileges and may contain bugs that an attacker can exploit. Many existing firewalls rely on approximate assessment of privileged systems software for their trustworthiness. This is sufficient if there are “well known working versions” of common programs such as the FTP server, *ftpd*. In some cases, however, the server can itself compromise security. A recent version of the WUArchive *ftpd*[2] contained a bug that permitted anyone on the Internet to gain super-user access to systems on which it was running. In our design, we attempt to sidestep the issue by providing proxies that can run locked into a specific subdirectory by means of “chroot” — a UNIX system call that permanently restricts the working filesystem of a process. Proxies are also designed to run without special system privileges, to further reduce the chance that they might be able to damage the system. Ideally it should be impossible for an outside user to ever interact with a privileged process. Practically speaking, the Internet service master daemon *inetd*, which is responsible for starting other service daemons, needs to run with privileges, but outside users cannot interact directly with it. There is a possibility that the kernel may have trapdoors or hidden network services built into it, but it is impractical to attempt to obtain and examine kernel sources for such flaws. Instead, make every effort to remove unnecessary kernel services at system build time.

Design Philosophy

The TIS Firewall Toolkit (hereafter referred to as “the toolkit”) is designed to be informally verified for correctness as a whole or at a component level. Since the firewall consists of discrete components, each providing a single service, each may be examined separately from the rest of the system. Components of the toolkit are as simple as possible in their implementation, and are distributed in source code form to encourage peer review. This appears to be a fairly novel approach for a network firewall, as many existing firewall systems rely on software that is "known to be good" or that is considered trustworthy because it has been used extensively for a long time.

One problem with the "known to be good" approach is that historically it hasn´t been very reliable. Certain software components are frequently exploited in break-ins, no matter how carefully they are maintained. Problem programs are usually complex pieces of software, implemented in tens of thousands of lines of code, which require system privileges in order to operate. As a step towards addressing this, the firewall toolkit operates in accordance with the following general firewall design principles:

- Even if there is a bug in the implementation of a network service, it should not be able to compromise the system. Services that are misconfigured should not work at all, rather than opening holes.
- Hosts on the untrusted network should not be able to connect directly to network services that are running with privileges.
- Network services are implemented with a minimum of features and complexity. The source code is simple enough to be reviewed thoroughly and quickly.
- There should be reasonable and pragmatic means of testing that the system is correctly installed.

The toolkit is designed to be used with a host-based security policy, but its components can be used with router-based firewalls. In this paper, we will focus on the former. In a host-based firewall, the security of the host is crucial; once it is compromised the entire network is open to attack. Still, we believe that a host-based firewall is superior to other solutions because of the ease with which it can be maintained, configured, customized and audited. When the toolkit is used with router-based firewalls, it is assumed that the toolkit software is running on a secure host that is permitted some degree of access between the protected network and the Internet, by means of routers. This leaves the option of configuring the routers to provide additional avenues between the protected network and the Internet for whatever reason; such additional avenues are outside the scope of the toolkit and should be provided only after careful security analysis.

The toolkit may be used in conjunction with router-based screening as extra security. To minimize risks, the services that are provided on the external machine, which we will refer to as a "bastion host", following the terminology proposed by Ranum[3]. are sharply curtailed and each service is subjected to review. On the "standard" firewall configuration, the only services supported are SMTP, FTP, NNTP, and TELNET. Other proxies such as Digital Equipment Corporation´s X Window System proxy [4] can be added to this architecture.

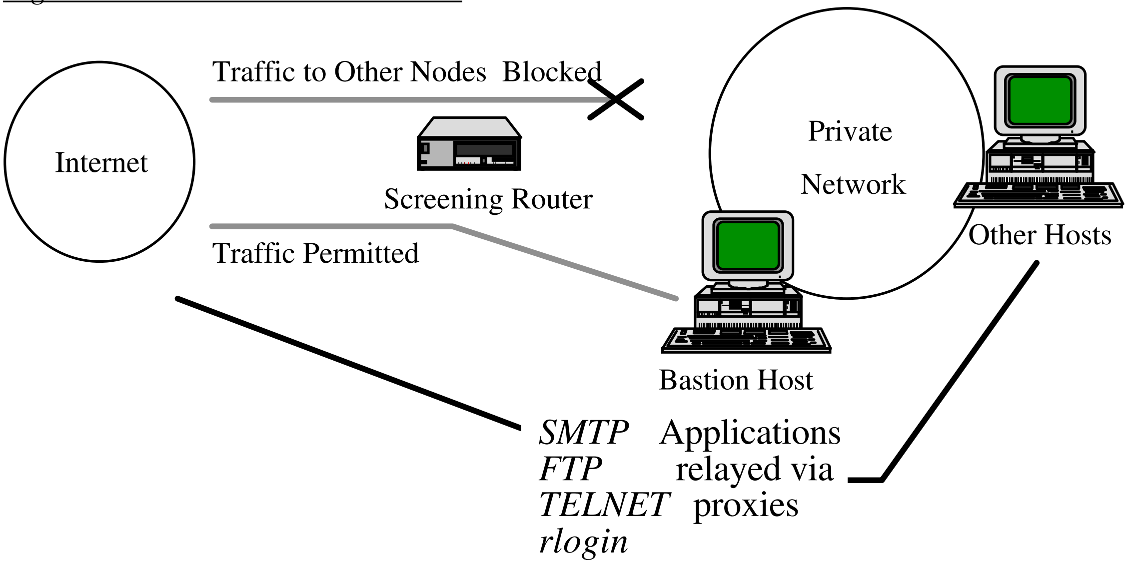
SMTP service is supported through a non-privileged front end that runs locked in a "safe directory" via chroot. FTP is supported via a proxy that runs without requiring special privileges. NNTP is supported via a "tunnel" server that permits traffic between a host on the inside and its news server on the outside. TELNET service is via a proxy that runs unprivileged. Since all other services on the system are disabled selectively, it is only these four services that must be analyzed for risk. By analyzing of the security of each service in isolation, we are able to gain a degree of trust in the system beyond merely being able to state "Well, we don´t *think* there are any bugs." With all the services running unprivileged we can make a stronger statement, to wit, "The security of an individual service is irrelevant to the overall security, as the server is running in a captive mode."

Configuration and Components

Figure 2 represents the toolkit installed in an environment that combines routers and a firewall bastion host. The implementation of the security controls is shared (in this example) between the routers and the firewall: the routers are responsible for controlling network-level access, and the bastion host provides application-level control. A simpler firewall configuration would consist of a dual-homed gateway, in which a workstation with two network interfaces is connected to both networks, and has IP forwarding disabled. Dual homed gateways are less flexible than firewalls that combine routers and hosts, since the option to route services at a network level is generally not available.[2] On the other hand, with a dual-homed gateway, the administrator can have a higher degree of confidence that no network traffic will be able to somehow "leak" through a router, since routers are no longer an integral part of the security system.

[2] Some versions of UNIX support packet screening within the operating system.

Figure 2: A Screened Host Firewall



The toolkit is designed to build a host-based firewall, with security being enforced by a single bastion host. For ease of management, all the proxies and access control tools use a single configuration file with a regular syntax. We thought this was useful due to the generally complex configuration of various publicly available firewall tools, of which no two are configured in the same way. The configuration rules are designed to provide both configuration and service and access permissions information, being read top-to-bottom and left-to-right. Hostnames or IP addresses including simple wildcards can be used in configuration rules, but IP addresses are preferred since DNS addresses are vulnerable to spoofing.

```
# Example ftp gateway rules:
# --------------------------
ftp-gw: authserver       127.0.0.1 7777
ftp-gw: denial-msg       /usr/local/etc/ftp-deny.txt
ftp-gw: welcome-msg      /usr/local/etc/ftp-welcome.txt
ftp-gw: help-msg         /usr/local/etc/ftp-help.txt
ftp-gw: timeout          3600
ftp-gw: permit-hosts     192.33.112.100
ftp-gw: deny-hosts       128.52.46.*
ftp-gw: permit-hosts     192.33.112.* -log { retr stor } -auth { stor }
ftp-gw: permit-hosts     * -authall
```

The firewall toolkit functionality can be broken down into six areas: logging, electronic mail, the Domain Name Service, FTP, TELNET, and TCP access control.

Logging

Significant security events and audit records are logged to a protected host on the internal network via the syslog facility. The version of *syslogd* that the toolkit uses is based on the BSD "net2" sources, with some modifications to support pattern-matching and program execution on matched patterns. Many systems administrators have batch processes set up on their systems to alert them of possible security problems by searching the system logs at regular intervals. By permitting the system manager to add regular expressions to the syslogd configuration, security-related log messages can be identified instantly. *Syslogd* contains further modifications that permit an arbitrary command to be invoked with any specified logging rule, so that, for example, vitally important security log events can be delivered to the system manager´s beeper or delivered by electronic mail. Adding command execution to *syslogd* implies that the *syslogd* configuration file must be protected against unauthorized modification.

Electronic Mail

Mailers are one of the favorite points of attack against UNIX systems. The Morris Internet worm exploited a well-known hole in the standard UNIX SMTP server, *sendmail*. Many systems running *sendmail*, including those with Internet firewalls, were penetrated by the worm. A few that had replaced *sendmail* with other SMTP servers were not. Since that time, a variety of other security holes have been identified in *sendmail* and fixed in more recent releases.

The problem with mailers is twofold: they are complex and perform file system activity, and they require privileges so that they can manipulate mailboxes or execute mail processing programs on the behalf of users. To help secure mail service, direct network access to sendmail is prevented. A simple program that implements a skeleton of the SMTP protocol is presented on the SMTP port on the mail server. This SMTP proxy, called *smap*, is small enough to be subjected to a code review for correctness (unlike *sendmail*) and simply accepts all incoming messages and writes them to disk in a spool area. Rather than running with permissions, the proxy runs with a restricted set of permissions and runs "chrooted" to the spool area. A second process is responsible for scanning the spool area and delivering the mail messages to the real *sendmail* for delivery — a mode of operation in which *sendmail* can operate with reduced permission. Many Internet firewalls run *sendmail* and rely on "trustworthy" versions of the software; running the mail software in a reduced-permissions mode is a more general solution to the problem, side-stepping the issue of whether or not a given version of *sendmail* contains bugs.

While *smap* answers all valid SMTP commands sent to it, it does not execute any of them except those directly involved with mail exchange: HELO, FROM, RCPT, DATA, and QUIT. Other commands, such as VRFY and EXPN return a polite error message. *Smap* preserves *sendmail´s* functionality, while preventing an arbitrary user on the network from communicating directly with it. Analyzing *sendmail´s* 20,000 lines of source code for bugs is a sizable task when compared to analyzing *smap´s* 700 lines. *Smap* is not a panacea, however, as firewalls remain vulnerable to data-driven attacks in which messages may be mailed to hosts on the private network, possibly triggering security holes in internal mailers. Since many of these attacks have a distinctive signature, *smap* or the firewall´s mailer can be configured to attempt to identify these letter-bombs, but the security administrator is forced into the unfortunate position of an arms-race in which a reactive role must be taken as new attacks are invented. To reduce the risk of attacks that exploit mailing through programs, the mailer on the firewall itself is configured so that program execution is disabled. Disabling program execution is often an unacceptable solution on a multi-user system, but since the firewall is not a general use host, we prefer to reduce the risk of someone being able to execute arbitrary commands from afar.

Domain Name Service (DNS)

The name service software available for UNIX implements an in-memory read-only database. As such, it cannot be used to gain unauthorized access to a system. Some past attacks on firewalls have used name service spoofing as a technique for impersonating trusted network hosts. In order to remove the threat of name service spoofing, the firewall does not rely on name service for any security related information. The name server software is necessary for high performance large-scale mail systems and is configured so that the only application that relies on name service for addressing is the electronic mail system. DNS names are also used in audit records, but are always presented along with host network addresses; mismatches are flagged as possible spoofing attempts.

FTP

The FTP application gateway is a single process that mediates FTP connections between two networks. Since it performs no disk access other than reading its configuration file and is a small and relatively uncomplicated program, it can be argued that it is not capable of compromising the security of the system. Just to be certain, the application gateway runs as a non-privileged user, after "chrooting" itself to a private directory on the system. To control FTP access, the application gateway reads a configuration file, containing a list of FTP commands that should be logged, and a description of what systems are allowed to engage in

FTP traffic. All traffic can be logged and summarized. Optionally, the gateway can permit FTP traffic from the Internet to the campus network for users who first authenticate themselves to the system.

TELNET

The TELNET application gateway is a small, simple application that mediates TELNET traffic. As with the FTP application gateway, the only file accessed is the configuration file that is read at start-up. Immediately after the configuration file is read, the TELNET application gateway is "chrooted" to a restricted directory, where it runs as a non-privileged process. The TELNET gateway´s configuration file allows specification of which systems or networks can use it, and what systems or networks it will permit connection to. Initially, it will be configured to permit campus systems to use the gateway to connect to Internet systems, but not vice-versa. Optionally, the TELNET gateway can require authentication before permitting use. All connections and their durations are logged.

UDP-Based Services

Since we decided that no direct traffic would be permitted between an outside system and an inside system, and since UDP is connectionless and point-to-point (and so cannot be used through network proxies), UDP services are not allowed. Many UDP-based services such as NTP and DNS can be provided transparently through a firewall by configuring the servers to act as forwarders for queries originating within the protected network.

TCP Access and Use

On BSD-based UNIX systems, most network processes are started up by an initial connection to a general-purpose network listener *inetd*, which establishes a connection between the incoming request and the program to service the request. For example, an incoming request for the TELNET service is "heard" by the running network listener. The program, according to *inetd´s* configuration file and the entry for TELNET, is executed and connected to the incoming request.

Inetd, the Internet services daemon, performs no function other than to invoke specified processes to manage network services when a system attempts to connect to them. Some vendor implementations permit a systems administrator to specify the user-id that the service should be invoked as, but there is no provision for limiting access based on the source of the request. A variety of implementations of "wrapper" processes are available on the Internet with varying functionality[5].

The toolkit uses a "wrapper" process called *netacl*, which provides support for all TCP-based services. (If only TCP-based services are supported, UDP services are disabled and are no longer a threat worth worrying about.) *Netacl* has no great advantages over other versions of TCP wrappers, other than its minimal size (240 lines of code, including a large copyright header and comments), its lack of support for UDP (purposely), and its sharing a common configuration mechanism with the other tools in the toolkit.

TCP Plug-Board Connection Server

Certain services such as Usenet news are often provided through a firewall. In such a situation, the administrator has the choice of either running the service on the firewall machine itself or installing a proxy server. Since running news on the firewall itself might expose the system to any bugs in the news software, it is safer to use a proxy to gateway the service onto a "safe" system on the campus network. *Plug-gw* is a general purpose proxy that "plugs" two services together transparently. Its primary use is for supporting Usenet news, but it can be employed as a general-purpose proxy if desired. *Plug-gw* is configurable, as are the other proxy servers. Since it only acts as a data pipe, it performs no local disk I/O and invokes no subshells or processes. Like the other proxy servers, it logs all connections.

Plug-boarding TCP connections through one´s firewall should be undertaken with a degree of caution, since plug-gw uses no authentication other than the host address of the client, and does no examination of the traffic passing across it. In the case of NNTP, for example, a security flaw in the NNTP server on the internal host could still be exploited. The firewall will make it much harder for an attacker to gain access to the internal system to further exploit the hole; if the flawed NNTP server were running on the firewall bastion host itself, the entire firewall might be vulnerable. Alternate approaches, such as engineering the news server to run "chrooted" are potential areas for future research. From a standpoint of systems administration, we have found that news administration is simplified by running it a readily accessible internal server.

User Authentication

The network authentication server *authsrv* provides a generic authentication service for toolkit proxies. Its use is optional, required only if the firewall FTP and TELNET proxies are configured to require authentication. *Authsrv* acts as a piece of "middleware" that integrates multiple forms of authentication, permitting an administrator to associate a preferred form of authentication with an individual user. This permits organizations that already provide users with authentication tokens to enable the same token for authenticating users to the firewall. A secondary goal of *authsrv* was to provide a simple programming interface for authentication service, since commercial authentication systems tend to have unique, nonstandard, interfaces. Several forms of challenge/response cards are supported, along with software-based one-time password systems, and plaintext passwords. Use of plaintext passwords over the internet is strongly discouraged, due to the threat of password sniffing attackers.

A simple administrative shell is included that permits the authentication database to be manipulated over a network, with optional support for encryption of authentication transactions. The *authsrv* database supports a basic form of group management; one or more users can be identified as the administrator of a group of users, and can add, delete, enable, or disable users within that group. *Authsrv* internally maintains information about the last time a user authenticated to the server and how many failed attempts have been made. It can automatically disable or time-lock accounts that have multiple failures. Extensive logs are maintained of all authsrv transactions. *Authsrv* is intended to run on a secured host, such as the bastion host itself, since its database must be protected from attack.

Testing Firewalls

Throughout the design of the toolkit, we tried to design each component so that it relied wherever possible on protections in the UNIX environment, rather than on elaborate code designed to check and deter threats. While the toolkit software doesn´t include a test suite, it is designed to be easy to verify that each component operates as it is intended. As an example, the SMTP proxy smap runs "chrooted" to a subdirectory as an unprivileged process. It stands to reason that if the proxy performs this operation properly, all files will be created in the proper directory, with the proper user permissions. If the administrator verifies that this is indeed the case, he can rely on the security of the operating system´s support for "chroot" and user file permissions. By examining the assumptions of each service proxy, a degree of assurance that the firewall is well protected can be gained. This does not address the problem of possible bugs or protocol errors in the proxy implementations that might still permit a service to pass through the firewall. To attempt to address this, every effort is made to keep the implementation of the proxies, especially the parts that deal with access control, as simple as possible.

Firewall administration requires a seasoned UNIX systems manager. While the toolkit is fairly easy to install, it assumes an amount of expertise on the part of the administrator, since he must know how to interpret error conditions, configure the system, and disable potentially threatening services. While it is a temptation to make the toolkit software self-installing and self-configuring, doing so raises the possibility that someone might install it who lacks the basic skills necessary to know if they have in fact secured their network. Packaging the toolkit as a set of components that can be used freely has proven effective, since it fills a need on the part of those experienced system managers who would have had to design, write, debug, and test their own implementations if ours were not available.

Future Directions

In the future we will focus on the problem of adding newer interactive information retrieval services such as Gopher, WAIS and World Wide Web and broadcast services such as MBONE. Possible avenues for future research include integrating cryptography with the firewall software to permit firewall-to-firewall service and firewall-to-firewall authentication, possibly using kerberos protocols. Support for IP-on-demand services like PPP pose a problem for firewalls: is the dial-up user to be treated as an untrusted Internet host or as a part of the protected network? Adding support for authenticated and encrypted PPP service on the firewall itself is being examined.

Observations

In practice, we find that running servers without special system privileges increases our assurance that the firewall is secure. More importantly, the methodology of turning off all services but a minimum, and then auditing each one on a case-by-case basis further increases confidence that the system is harder to break into. The basic design decisions in setting up a firewall (to route or not to route, to rely on the host or the router) remain unchanged, but the toolkit will work with either model.

Firewalls are a stop-gap measure that is needed because many services are developed that operate either with poor security or no security at all. Perhaps the most important lesson we can learn from firewalls is the need for strong session-level authentication in applications and well-designed application protocols.

Availability

The TIS Internet Firewall Toolkit is available in source form via anonymous FTP from ftp.tis.com: /pub/firewall/toolkit/fwtk.tar.Z. Information is available from the authors at *fwall-support@tis.com*. Send mail to *fwall-users-request@tis.com* to be added to the firewall toolkit user´s mailing list. Future enhancements to the toolkit will be announced on *fwall-users* and other relevant mailing lists.

Acknowledgements

This work was done, in part, under a contract from the U. S. Department of Defense, Advanced Research Projects Agency (ARPA), number DABT 63-92-C-0020. [6]

References

[1] Marcus J. Ranum, "Thinking About Firewalls," Proceedings of Second International Conference on Systems and Network Security and Management (SANS-II), April, 1993

[2] Washington University Saint Louis, FTP server daemon. Available for FTP from wuarchive.wustl.edu

[3] Marcus J. Ranum — "An Internet Firewall," Proceedings of First International Conference on Systems and Network Security and Management (SANS-I), Nov, 1992

[4] G. Winfield Treese and Alec Wolman, "X Through the Firewall, and Other Application Relays," Proceedings of USENIX Summer Conference, 1993. Also available as Cambridge Research Lab Technical Report 93/10, Digital Equipment Corporation, May 3, 1993.

[5] Wietse Venema, "TCP WRAPPER, network monitoring, access control, and booby traps," UNIX Security Symposium III Proceedings (Baltimore), September 1992

[6] Frederick M. Avolio and Marcus J. Ranum, "A Network Perimeter With Secure External Access," Internet Society Symposium on Network and Distributed Systems Security, February 1994.

William Cheswick, "The Design Of a Secure Internet Gateway," Proceedings of the 3rd USENIX Security Symposium, September 1992.

Stephen M. Bellovin and William Cheswick, "Firewalls and Internet Security: Repelling the Wily Hacker," Addison-Wesley, Spring 1994

Frederick M. Avolio is a principal analyst with Trusted Information Systems, Incorporated, and active in network security consulting and product development. He has lectured on the subject of Internet gateways and firewalls and electronic mail configuration and has performed consulting services in these areas, both for government and in the private sector. He has worked in the UNIX and TCP/IP communities since 1979.

Mr. Avolio has an undergraduate degree in Computer Science from the University of Dayton and a Master of Science from Indiana University.

Marcus Ranum is a senior scientist at Trusted Information Systems. He is the chief architect of the firewall toolkit and spends most of his time on Internet security issues.