TIS Firewall Toolkit

Configuration and Administration

About this Documentation

The TIS firewall toolkit is not a single, integrated package — rather, it is a set of tools for building a variety of types of firewalls. As such, it is designed to be flexible in use and installation, which makes it difficult to document in a cookbook manner. This documentation describes in general terms how the firewall toolkit components might be installed on a typical system. Specifics of how the toolkit components function are documented in their respective manual pages. This document, the administrative overview, is intended for users who are interested in installing the software, and is written with the assumption that the reader is familiar with UNIX systems management. Generalities of UNIX systems administration and TCP/IP networking are outside of the scope of this documentation, and are a prerequisite to using this toolkit. Since the firewall being built can take several forms (see the *Overview*) and can use several different types of platforms, certain basic operations will be referred to in a very general manner. For example, if the firewall is being built with a router to provide packet screening, no attempt will be made to describe how to configure the router other than, “Set up the router's screening.”

An important aspect of setting up a firewall is assurance; knowing each step has been performed correctly provides a solid basis before proceeding to the next step. Wherever appropriate, we will offer suggestions for verifying the correctness of the firewall as it is being assembled.

Initial Configuration of the Network

Since the firewall software is all host-based, we can begin by assuming that the network is set up following whatever policy has been elected to prevent unwanted traffic between the protected and untrusted network. This can be done either via a screening router or implicitly by using a dual-homed gateway. Often, if network service is provided by a network carrier, they will be willing to configure security in the router when the link is set up. Even if using a dual-homed gateway, there is no reason not to take advantage of the router, to further pre-screen traffic, if it is capable of doing so.

Version Dated: 02/17/94

A word of warning: during this brief time, the firewall bastion host may be exposed to attack, before its security has been configured. It is worth the few minutes required to temporarily edit inetd.conf to disable incoming network services, and to use the console while performing the initial set-up.

At this point, we can postulate that traffic between the untrusted network and the protected network is, in fact, blocked according to whatever policy has been selected. For the sake of argument, we'll assume that all traffic is to be blocked in either direction. This hypothesis should be empirically tested before proceeding. If desired, there is a program within the tools/admin/netscan directory of the toolkit distribution, which will attempt to connect to all the hosts on a specified subnet. This can be used to attempt to probe through the firewall in either direction, to make sure that traffic is indeed blocked. Since blocking traffic is the linchpin of the firewall's security, do not proceed until confident that it's performing properly.

Initial Configuration of the Bastion Host

One of the basic principles of building a firewall is to shut down all unknown and unnecessary services. Shutting down unnecessary services on the bastion host requires some system-specific knowledge. Generally, this step entails:

- Editing /etc/inetd.conf
- Editing system startup scripts (/etc/rc, and /etc/rc.local, etc.)
- Editing the operating system configuration to disable undesirable kernel-based network services, such as NFS, and rebuilding the kernel

This process should continue until the system is reduced to a minimum of services that the administrator is willing to trust — which should be a *short* list. Performing this configuration should be done concurrently with checking to see what services are actually running. Fortunately, most UNIX systems provide good tools for testing, such as ps and netstat. On a firewall in a reduced configuration, the system should look pretty quiet:

```
% ps -aux
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
mjr       3839 30.8  4.1  168  472 p0 R    14:27   0:00 ps -aux
root         1  0.0  0.3   56   40 ?  S    Aug 27  0:14 /sbin/init -
root        66  0.0  0.1   24   16 ?  S    Aug 27 37:55 update
root         2  0.0  0.0    0    0 ?  D    Aug 27  0:01 pagedaemon
root         0  0.0  0.0    0    0 ?  D    Aug 27  0:14 swapper
root     25081  0.0  0.0   56    0 co IW   Sep  1  0:00 - std.9600 console (gett
root        71  0.0  0.0   56    0 ?  IW   Aug 27  0:59 inetd
root      3812  0.0  0.6   64   64 ?  S    14:24   0:00 /usr/etc/in.telnetd
root     14410  0.0  0.8   64   96 ?  S    Aug 30  1:02 syslogd
root        69  0.0  0.0   88    0 ?  IW   Aug 27  1:00 cron
mjr       3813  0.0  1.5   64  168 p0 S    14:24   0:00 -csh (csh)
%
```

There are a bare minimum of processes running, and a bare minimum of network services available:

```
% netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  *.chargen              *.*                    LISTEN
tcp        0      0  *.daytime              *.*                    LISTEN
tcp        0      0  *.discard              *.*                    LISTEN
tcp        0      0  *.echo                 *.*                    LISTEN
tcp        0      0  *.time                 *.*                    LISTEN
tcp        0      0  *.smtp                 *.*                    LISTEN
tcp        0      0  *.finger               *.*                    LISTEN
tcp        0      0  *.telnet               *.*                    LISTEN
tcp        0      0  *.ftp                  *.*                    LISTEN
udp        0      0  *.syslog               *.*
udp        0      0  *.chargen              *.*
udp        0      0  *.daytime              *.*
udp        0      0  *.discard              *.*
udp        0      0  *.echo                 *.*
udp        0      0  *.time                 *.*
udp        0      0  *.1030                 *.*
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Vnode     Conn     Refs  Nextref Addr
 f70930c dgram       0      0        0        0        0        0
 f70808c dgram       0      0  f0cbc48        0        0        0 /dev/log
%
```

Most of the servers configured are configured with some form of additional security, which will be described later. A general purpose tool for probing for TCP-based services is included with toolkit in tools/admin/portscan, which can be used to probe a system to see what TCP services it is providing.

Installing the Toolkit Components

The toolkit software contains instructions and makefiles for compiling and installing the components (see the “README” file in the top-level directory). By default, the toolkit components will all install in /usr/local/etc, where they will not be overwritten during operating system upgrades or patch installation. Certain of the extra components, such as the enhanced syslogd daemon, would replace standard system binaries. It is recommended that modified system files be renamed and replaced with a symbolic link, if appropriate.

All the toolkit components that require configuration information share a common configuration file (by default /usr/local/etc/netperm-table). Most of the toolkit components are invoked via inetd. For more detailed information on installing each component individually, see the manual page for that component. A sample /etc/inetd.conf file might resemble:

```
#
ftp     stream  tcp   nowait  root    /usr/local/etc/netacl   in.ftpd
ftp-gw  stream  tcp   nowait  root    /usr/local/etc/ftp-gw   ftp-gw
telnet-a stream tcp   nowait  root    /usr/local/etc/netacl   in.telnetd
telnet  stream  tcp   nowait  root    /usr/local/etc/tn-gw    tn-gw
login  stream   tcp   nowait  root    /usr/local/etc/rlogin-gw rlogin-gw
finger  stream  tcp   nowait  nobody  /usr/local/etc/netacl   in.fingerd
smtp    stream  tcp   nowait  root    /usr/local/etc/smap     smap
# Time service is used for clock syncronization by folks to lazy to use NTP
time    stream  tcp   nowait  root    internal
time    dgram   udp   wait    root    internal
echo    stream  tcp   nowait  root    internal
echo    dgram   udp   wait    root    internal
discard stream  tcp   nowait  root    internal
discard dgram   udp   wait    root    internal
daytime stream  tcp   nowait  root    internal
daytime dgram   udp   wait    root    internal
chargen stream  tcp   nowait  root    internal
chargen dgram   udp   wait    root    internal
```

The netacl program is a “TCP wrapper” program that provides access control for TCP based services. It is similar to other published TCP wrappers such as log_tcp, but is included with the toolkit because it is a minimal implementation (therefore easier to understand and trust) and uses the same configuration file as the other tools in the toolkit.

When setting up the toolkit tools, a “locked room” directory should be configured on the system. This directory is where tools that perform a chroot(2) will isolate themselves. The locked room directory should not be the same directory as is used for anonymous FTP, if that service is provided. Depending on the library support on the system, and name resolution support, it is possible that the locked room directory will require a /etc directory with a resolv.conf file or similar support files. Do not, under any circumstances, install copies of the password file, executable images, or device nodes in the locked room directory.

Generally, the first step is to configure netacl to permit internal systems limited access to the firewall, if desired for administrative purposes. Depending on whether the TELNET gateway tn-gw is employed, administrative access to the firewall may have to be on a different port than the standard TELNET port (23). This is because the telnet program often disables options processing when connecting to any port other than the standard port — so the proxy service must run on the TELNET port, and the actual service on a different port (in the example inetd.conf file above, the service is named telnet-a) Verify that netacl is performing properly by configuring some specific permit/deny rules for individual hosts, and then attempt to access services from them.

Once netacl is configured, the TELNET and FTP gateways can be configured. When configuring the TELNET gateway, simply install it as the service executable in inetd.conf and write some permissions rules describing what systems can employ it. Optionally, help and banner files can be defined, which are presented to the user when requested. Configuring the FTP proxy is similar in practice. If the bastion host is currently supporting FTP service, the FTP proxy may be configured to run on another service port. Many client implementations of ftp are capable of using a non-standard service port, unlike telnet. Banner, help, and service-denied message files may be tailored to suit.

The rlogin proxy is optional, and, to be used, must be installed on the bastion host's rlogin service port (port 512). Rlogin's protocol requires binding a “privileged” port, an operation that requires system permissions under UNIX. Administrators wishing strict security should set the proxy's *directory* option to ensure it runs under chroot.

The smap and smapd sendmail wrapper processes may be installed, using the locked room directory for spooling, or using a different directory elsewhere on the system. Smap/smapd do not replace sendmail, so it will still be necessary to configure sendmail for the firewall, which is outside of the scope of this documentation.

Establishing a Permissions Set

When configuring proxy servers and network access control programs, it is important to establish a permissions set that accurately reflects the security model that is desired. A good place to start is by configuring the firewall so that anyone on the “inside” can use the proxies freely, while nobody on the “outside” can use them at all. Setting this up in the firewall configuration file is quite straightforward, as it was designed to most easily support the basic case. A sample configuration for the FTP proxy is:

```
# Example ftp gateway rules:
# -------------------------
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:        permit-hosts 192.33.112.* -log { retr stor }
ftp-gw:        timeout 3600
```

In this example, hosts on network 192.33.112 are permitted to use the proxy. Since no other hosts are listed, all other access is disabled. If a host from a different network attempts to use the proxy, they will be displayed the contents of the *denial-msg*, in this example /usr/local/etc/ftp-deny.txt, and disconnected. Later, if the protected network grows and additional subnets are added, they can be either added to the single “permit-hosts” line, or a second line can be added:

```
ftp-gw:         permit-hosts 192.33.112.*  16.67.32.* -log { retr stor }
```

0r:

```
ftp-gw:         permit-hosts 192.33.112.* -log { retr stor }
ftp-gw:         permit-hosts 16.67.32.* -log { retr stor }
```

For a more detailed description of the syntax of the netperm-table file, refer to the manual page that is included with the toolkit software. The syntax of the permission file is documented in the netperm-table manual page. Each firewall firewall component may have component-specific options and flags that it can be configured with. Component-specific options and rules are described in the component's manual page. In the example above, special options to the FTP proxy are provided to specify that logging for the *retr* and *stor* FTP options should be enabled.

Anonymous FTP

Anonymous FTP servers have been available on UNIX systems for over 10 years. Security holes in anonymous FTP still crop up occasionally, either because new functionality is added and bugs are introduced, or because systems are misconfigured. One approach to securing anonymous FTP is to use netacl as a means of ensuring that the FTP server process is chrooted *before* it is invoked. In such a configuration, it is extremely difficult for a security hole in the FTP server to compromise the system outside of the FTP area. An example configuration is using netacl to perform or not perform the chroot depending on the origin of the connection. Suppose the protected network is network 192.5.12 netacl may be configured as followed:

```
netacl-in.ftpd hosts 192.5.12.*  -exec /usr/etc/in.ftpd
netacl-in.ftpd hosts unknown     -exec /bin/cat /usr/local/etc/noftp.txt
netacl-in.ftpd hosts *           -chroot /var/ftpdir -exec /bin/ftpd
```

In this example, users connecting to the FTP service from the protected network have normal FTP capabilities. Users connecting from systems that are not in the DNS are presented a message informing them that they are not permitted to use FTP. All other systems connecting to the FTP service will be using a version that has been pre-chrooted to the FTP file area. This has several advantages in addition to security. Firstly, when ftpd checks for user authentication, it will read the /etc/passwd file from the FTP area, permitting administrators to give out "accounts" for the FTP area itself[1]. It is possible in this manner to have an FTP area that is maintained by users who have no accounts on the bastion host, while gaining excellent audit trail and authentication. It also permits the administrator to use the more feature-rich implementations of ftpd, which may contain security bugs. After all, if a program is widely used for over 10 years and consistently causes security problems, it is a leap of faith to assume that any given version is trustworthy.

Supporting both FTP and the FTP proxy on one host

The FTP proxy may be invoked either by running it on a dedicated port or by running it from within a modified FTP daemon. If the FTP proxy is run on the FTP service port, then the host providing the FTP service cannot also act as an FTP server. In order to resolve this conflict, the toolkit includes a version of the FTP daemon that “understands” the existence of the proxy and invokes it when it recognizes an address in the form of *user@host*. The source code to the modified FTP daemon are included in the toolkit directory tools/server/ftpd and must be compiled with the option PROXY_PASSTHROUGH defined in the Makefile. This option should be defined as the pathname of the proxy executable. If using the modified FTP daemon in conjunction with the chroot-before-invocation approach described above, the pathname for the proxy server should be relative to the FTP directory area (e.g, /bin/ftp-gw) rather than the root filesystem. When using

[1] Care should be taken to prevent guessable passwords in the FTP area's passwd file, to modify ftpd to check the inode on the password file and prevent retrieval of it, or to use a version of ftpd that uses the authentication server.

chrooted ftpd, a duplicate copy of the ftp-gw and ftpd specific rules from netperm-table must be installed in the FTP area (e.g., in ~ftp/usr/local/etc/netperm-table).

Telnet and Rlogin

Generally, access to the bastion host should be restricted only to administrative logins. In order to run the proxies, normal telnet and rlogin service on the bastion cannot be provided on the standard service ports. There are three approaches to solving this problem:

- Run the telnet and rlogin proxies on the standard service ports, with telnet and rlogin servers on an alternate port, and protect access to them with netacl.

- Permit login only via the console.

- Use netacl to "switch" services depending on the origin, so that users who connect to "localhost" via the proxy get the real telnet service.

The latter solution is very convenient but permits anyone who is permitted to use the proxy to attempt to login to the bastion host. If the bastion host employs strong authentication (*highly* recommended) to control user access, the risk of attack via this route is minimized. To configure the system in this manner, first mediate all access to the login services with netacl, and have it invoke either the standard server or the proxy server depending on origin:

```
# myaddress below should be the bastion host's IP address
netacl-in.telnetd:   permit-hosts 127.0.0.1 -exec /usr/etc/in.telnetd
netacl-in.telnetd:   permit-hosts myaddress -exec /usr/etc/in.telnetd
netacl-in.telnetd:   permit-hosts * -exec /usr/local/etc/tn-gw
netacl-in.rlogind:   permit-hosts 127.0.0.1 -exec /usr/etc/in.rlogind
netacl-in.rlogind:   permit-hosts myaddress -exec /usr/etc/in.rlogind
netacl-in.rlogind:   permit-hosts * -exec /usr/local/etc/rlogin-gw
```

Administrators wishing login access to the bastion host must first connect to the proxy and then command it to connect them to the bastion host. This works nicely because some versions of the telnet and rlogin clients will not work properly if connecting to a non-standard service port.

Authentication and the Authentication Server

The toolkit includes an optional authentication server, authsrv, which is designed to support multiple authentication mechanisms in a mechanism-independent manner. Authsrv maintains an internal user database with a record for each user including the authentication mechanism to use for that user, the user's group, long name, last successful authentication, etc. Plaintext passwords for local use are supported within the internal database, for situations where an administrator wishes to control access to the firewall services by users on the protected network in a simple cost effective manner. *Plaintext passwords should not be used for authentication by users on untrusted networks*. Authsrv should be run on a host that is as secure as possible, generally the bastion host itself. To ease management of

authsrv, a networked administrative shell, authmgr, can be used to manipulate the authsrv database remotely, with optional encryption to protect the communication.

Users in the authsrv database can be divided into different control groups, managed by a group administrator. When a group administrator is created, that administrator has complete control over the group, including the ability to add or delete users from the group. This functionality is present to simplify administration where a single firewall is being used by multiple organizations.

Authsrv contains support for 5 forms of authentication at initial release. These are: internal plaintext passwords, Bellcore's S/Key[2], Security Dynamics' SecurID, Enigma Logics' Silver Card, and Digital Pathways' SNK004 Secure Net Key. When authsrv is compiled, the administrator should edit the authentication protocol bindings in auth.h to reflect the forms of authentication in use locally. As users are created, the administrator must specify the type of authentication protocol that should be applied to the user. Typically, a mix of authentication systems will be employed, based on considerations such as cost, existing availability, and ease of use.

Support for authentication via authsrv is present in all the interactive proxies included with the toolkit, and may be enabled by specifying options in the configuration file. Authentication can be specified directionally, e.g.: “incoming must authenticate, outgoing requires no authentication.” Authsrv uses a simple challenge/response protocol internally, which is intended to be easily integrated into other software.

To configure authsrv, identify a free TCP service port number and add an entry to inetd.conf to invoke authsrv each time there is a connection request. Authsrv is not a daemon process that stays running, it is a program invoked once per request, and due care is paid to ensure that it locks its database to prevent corruption. Adding authsrv to inetd.conf may require creating a corresponding entry in /etc/services. Since authsrv accepts no command line parameters, the inetd.conf and services entries might resemble:

In /etc/services:

```
authsrv     7777/tcp
```

*In /etc/inetd.conf: (**Warning**: Some versions of UNIX use different syntax)*

```
authsrv stream tcp   nowait  root    /usr/local/etc/authsrv authsrv
```

The service port chosen should be used to configure the client applications that will be using authentication. Configuring a client application to use authsrv entails making an entry in the client's configuration information, and configuring the client to require authentication for some service. Authentication need not be required for all operations of a client, e.g.:

---

[2] S/Key is available for FTP from thumper.bellcore.com, in pub/nmh/skey. The toolkit does not include the complete S/key software.

```
# Example ftp gateway rules:
# -------------------------
ftp-gw:   authserver      localhost 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:   permit-hosts    192.33.112.100
ftp-gw:   permit-hosts    192.33.112.* -log { retr stor } -auth { stor }
ftp-gw:   permit-hosts    * -authall
ftp-gw:   timeout         3600
```

In the example above, authentication is being used with the FTP proxy. The first entry defines the network address and service port number of the authentication server. The *permit-hosts* entries show some of the flexibility of the authentication system. One host has been chosen (for some reason unknown to us) to require no authentication. If a user connects to the FTP proxy from that host they will be able to use all of the proxy's services freely. The second *permit-hosts* rule requires authentication for all systems in network 192.33.112 that wish to export files. This is implemented by the optional clause:

```
-auth { store }
```

Which specifies a list of FTP operations that will be blocked until the user has authenticated successfully to the server. When a user authenticates successfully, the commands are unlocked, and the fact that the user authenticated successfully is logged. The final example *permit-hosts* rule specifies that any other system may communicate with the server, but in order to perform *any* FTP operation, the user must authenticate first.

The authsrv server in turn must be configured to recognize which clients it will permit to connect with it. This is to prevent unwanted attempts to probe the server by hosts that are not running software that requires authentication. Typically, in a firewall, the authsrv will be run on the bastion host, along with the proxies that rely on it. If no other systems require access to the authsrv, then both client applications and the server should use “localhost” as the communications address. The authsrv configuration rules specify where it should maintain its database, and the clients it supports:

```
# Example authsrv rules:
# ---------------------
authsrv:   database      /var/adm/authsrv.db
authsrv:   permit-hosts  localhost
authsrv:   permit-hosts  192.5.214.32 cipherkey
```

In the example above, the pathname of the database is specified, and two client hosts are recognized. Note that the database should be on a protected system, or it should be protected carefully with file access permissions. Protecting the database is of utmost importance, which is why it is recommended that the database reside on the bastion host. The second client entry is an example of a client that will use DES encrypted communications while communicating with the authsrv. The cipher key is contained in the configuration file, requiring that the configuration file be protected from casual observation. Generally, encryption need not be used. The purpose of the encryption support is to permit an administrator to manage the authentication database from a personal workstation, rather than having to log into the bastion host. The only component of the traffic which needs to be protected is if the administrator is setting user's passwords over the local network, or if the administrator is managing a database over a wide area

network that may be tapped. Support for the encryption functionality is optional, requiring that two source code modules be replaced, and that a compatible DES encryption library be provided.

Maintaining the database is performed using two tools: authload and authdump, which respectively load or dump the authentication database. Administrators may wish to put an invocation of authdump in crontab, to make an ASCII backup copy of the database in case of corruption or accidental deletion. Authload is useful for bulk-loading user entries, or for initializing or exchanging the database.

Authload operates on individual records, and does not truncate an existing database. Therefore, it can be used to bulk-load new entries in the database, or to share databases between sites. Existing records will be overwritten if newer versions are loaded by authload. This permits an administrative record to be created if needed. Technically, there need not be an administrative record in the database for normal operations. If authsrv is invoked by user-id zero at the command line, it reverts to a special privileged administrative mode. In this mode, database-wide administrative operations may be performed with privileges.

Authsrv has a very flexible notion of groups and group management. The administrator can assign users to groups, where a group consists of an arbitrary short string of text. Group administration privileges may be granted with the “groupwiz” command. Once a user has group administrative privilege, they can create or delete records within that group, view records, enable and disable user records, and change their passwords. Group administrators may not create new groups or change group membership, effectively compartmenting the group administrator's powers to operations only within their own group. Group management may prove a useful capability for organizations in which there are several subgroups using the same firewall. Each group can be separately administered in a way that cannot interfere with others, outside of the requirement that user id names remain unique.

Creating a user is performed with the “adduser” command:

```
adduser mjr 'Marcus J. Ranum'
```

When a user record is first created, it is disabled, and the user still may not log in. Before a user may log in, the administrator may wish to set a password on the record, and may wish to change the user's group identity.

```
group users mjr
password "whumpus" mjr
proto SecurID mjr
enable mjr
```

When a user record is created by a group administrator, the user's group is automatically inherited, as is the authentication protocol to use. User records may be displayed using the “display” command. The “list” command may be used to display a summary listing of the entire database, or the group administrator's group, depending on the permissions of the invoker.

A Sample Authmgr Session:

```
%-> authmgr
Connected to server
authmgr-> login
Username: wizard
Challenge "200850": 182312
Logged in
authmgr-> disp wizard
Report for user wizard (Auth DBA)
Last authenticated: Fri Oct  8 17:11:07 1993
Authentication protocol: Snk
Flags: WIZARD
authmgr-> list
Report for users in database
user    grou    longname      flags  proto    last
----    -----   --------        ---  -----    ----
wizard          Auth DBA        y W  Snk      Fri Oct  8 17:02:56 1993
avolio  users   Fred Avolio     y    passw    Fri Sep 24 10:52:14 1993
rnj     users   Robert N. Jesse y    passw    Wed Sep 29 18:35:45 1993
mjr     users   Marcus J. Ranum y    none     Fri Oct  8 17:02:10 1993
authmgr-> adduser dalva 'Dave Dalva'
ok - user added initially disabled
authmgr-> ena dalva
enabled
authmgr-> group dalva users
set group
authmgr-> proto dalva Skey
changed
authmgr-> disp dalva
Report for user dalva, group users (Dave Dalva)
Authentication protocol: Skey
Flags:none
authmgr-> password dalva
Password: ########
Repeat Password: ########
ID dalva s/key is 999 sol32
authmgr-> quit
%->
```

In the example presented above, an administrator connects to the authsrv over a network using the authmgr interface. After authenticating as "wizard" (a privileged user), the "wizard" user record is displayed, showing the last authentication time, and authentication protocol. Note that the user was presented as Digital Pathways SNK prompt at login, as appropriate for the authentication protocol defined for the "wizard" user. Next, a listing of the users in the database is retrieved. A new user is added, with the adduser command, then enabled and assigned to a group and authentication protocol. The user's password is then set. Note that when using the authmgr interface, passwords are gathered without echoing them to the terminal. When setting passwords, it is best to use authmgr rather than authsrv in administrator mode.

Initializing an Authsrv Database

```
#
# authsrv

-administrator mode-
authsrv# list
Report for users in database
user   group   longname      flags  proto    last
----   -----   --------        ---  -----    ----
authsrv# adduser admin 'Auth DBA'
ok - user added initially disabled
authsrv# ena admin
enabled
authsrv# superwiz admin
set wizard
authsrv# proto admin Snk
changed
authsrv# pass '160 270 203 065 022 034 232 162' admin
Secret key changed
authsrv# list
Report for users in database
user   group   longname      flags  proto    last
----   -----   --------        ---  -----    ----
admin          Auth DBA        y W  Snk      never
authsrv# quit
#
```

In the example above, an empty database is initialized with the administrator's record. The administrator's record is enabled, granted administrative privileges, and assigned an authentication protocol. This example shows the authentication protocol selected as 'Snk', the SecureNet Key from Digital Pathways. The password value that is set is the shared secret key used by the SNK, a 24 digit DES key. Ensure that the permissions on the database file are such that it is protected against casual perusal.

Users and Changing Passwords

While the use of plaintext passwords is discouraged, some authentication systems have a form of password information that is settable per user (e.g.: SNK keys, SecurID PINs, and S/Key hash words). These values can be set by the administrator, but sometimes must be set by users, as in the case of S/Key hash words. In order to permit users to set their own secret values, the telnet and rlogin proxies have a built-in password command that prompts the user for a username and password, then prompts them for a new password. In order to prevent users from resetting their passwords across untrusted networks, the proxies have a special configuration option ("-pwok") that can be specified to indicate that a given set of systems is trustworthy enough that users may change their passwords if connected from them. Generally, these systems should all be on the internal network.

The authentication manager is not intended to address the problem of authentication on an untrusted LAN. Its purpose is to provide an easy interface for strong authentication for firewalls and basic network services. The capability of authenticating users with plaintext passwords is included for convenience and audit capability, and should not be used to mediate access to the firewall from untrusted networks. Authentication using authsrv and challenge-response calculators should provide protection adequate for authentication over untrusted networks. Functionality, such as the ability to remotely manage the database using authmgr is provided for convenience, and should only be used

after the administrator has carefully considered the degree of local threat that is being protected against, and the security of the system from which management is performed. If at all possible, the administrator and group administrator should use a strong form of authentication as their protocols, and not simple plaintext passwords.

Debugging the Firewall Toolkit Components

When installing and testing the firewall components, the administrator should monitor the system log maintained by syslogd. Many of the toolkit components are designed to run under chroot, and assume that there is no other means of logging errors and warnings, and leave diagnostic information *only* in the system log. Every effort has been made to make the diagnostic messages helpful and self-explanatory.

System Upgrades and Management

Systems maintenance and management are local policy decisions, but in general, with a network firewall, “if it isn't broken, don't fix it” is the best policy. As long as the base platform that the bastion host is running is reasonably stable, there should be no need to perform a complete system upgrade. Generally, upgrades are performed either to fix bugs in user applications, or to add new features. Since it is recommended that the firewall bastion host have no user accounts on it, non-critical bugs don't require an upgrade. In general, if the firewall works when it is first installed, it should continue to work, and there's no reason to interfere with it unless it suddenly ceases to work.

Managing the bastion host is a simple UNIX systems management problem, possibly made even simpler by the lack of user accounts. Tools such as watcher[3] and COPS[4] are not included with the firewall toolkit, but might be worth installing to provide warnings of system problems or possible security breaks. The toolkit includes several scripts in tools/admin/reporting which can be installed in crontab to generate automatic summaries of traffic through the firewall. A typical way of installing the software is to run a weekly report of the firewall's activity, which is emailed to the administrator(s), and a nightly or daily summary of security-related information.

Bugs and Problem Reporting

The firewall toolkit is intended to be a living body of software, with bug fixes (and possibly a few enhancements) being released periodically. If a bug is identified in a component of the software, send Email to *fwall-support@tis.com*[5] containing as much

---

[3] Watcher is a tool that monitors a system and alerts the systems administrator in the event that disk drives begin to fill up, or processes use too much CPU. Watcher can be FTPed from servers on the Internet, such as gatekeeper.dec.com: */pub/usenet/comp.sources.unix/volume11/watcher*

[4] COPS is a tool that scans a system for well-known security holes. While it's a reactive tool, and cannot protect against unknown or new holes, it is useful as a sanity check. COPS can be FTPed from gatekeeper.dec.com: */pub/usenet/comp.sources.unix/volume21/cops*

[5] *Fwall-support* is not a funded service - response to queries is on a “best effort” basis and may take a day or two.

detail as possible about the bug, including how to duplicate it, if that is known. In the event of a security-related bug being identified in the firewall toolkit, the presence of a fix will be announced on the internet firewalls mailing list *firewalls@greatcircle.com* after known users are notified. To subscribe to the firewall toolkit user's group mailing list, send Email to *fwall-users-request@tis.com.* New functionality may periodically be added to the toolkit, in the form of new applications and proxies[6]. Contrary to the normal evolution of a software system, an effort will be made to limit the adding of new functionality to the existing software base, unless it enhances security. Functionality for its own sake reduces security, and the firewall toolkit is, first and foremost, security software.

[6] Contact fwtk-bugs if you have proxies or applications you wish to donate.