SOCKS

David Koblas
Independent Consultant[1]
koblas@sgi.com

Michelle R. Koblas
Computer Sciences Corporation
NASA Ames Research Center
mkoblas@nas.nasa.gov

Abstract

This paper presents the Socks package, an Internet socket service consisting of client library routines and a daemon which interact through a simple protocol to provide convenient and secure network connectivity through a firewall host. Client software applications can be easily modified to utilize the Socks library routines in place of the normal socket library calls such that all outgoing connections will go through the Socks daemon (sockd) running on the firewall host. We will review several methods for setting up secure environments and then explain the detailed mechanisms of the Socks package. A current implementation will also be briefly discussed along with experiences with it.

1.0 Introduction

Security is a major consideration when connecting a network to the Internet. One of the more important issues which must be addressed is intruders attempting to gain access to local hosts. A common method for preventing these types of intrusions is to install a "firewall", a single point of attachment to the Internet which can be made highly secure. This paper presents the Socks library and daemon package. Using this package in conjunction with a network application (such as ftp) allows users convenient access to the resources of the Internet through a firewall hosts, while preventing unwanted intrusion. Although there are several possibilities for the setup of a firewall, the Socks package presents a simple, vendor-independent and unique solution which poses the least inconvenience for local users and maintains the integrity of the firewall.

1.1 Potential Solutions

This section will briefly review several strategies which can be used to configure an Internet connection to prevent unwanted intrusion and the advantages and disadvantages of each. The following solutions are presented:

- Having two sets of hosts -- secure (isolated) and non-secure (those connected to the Internet).

---

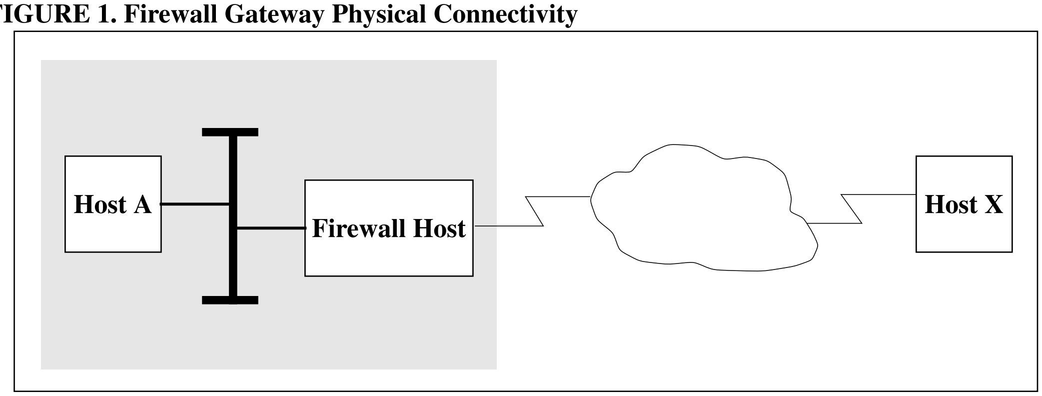
1. Developed while employed at MIPS Computer Systems, Inc.

- Setting up a firewall host which users have accounts on and allowing traffic to and from this host, but not allowing any traffic to pass through it.
- Utilizing router filtering such that only certain hosts/ports can connect to the firewall from the external network.
- Setting up a firewall host which uses the Socks package such that users are not required to have accounts on this host.

The most simple and obvious method for providing a completely secure environment is to have two sets of hosts: secure and non-secure. Secure hosts have no Internet access and operate only on an isolated network within their environment. Non-secure hosts are those which are connected to the Internet and communication between these hosts and secure hosts must be done manually (e.g. via tape). This method has the disadvantage of being cumbersome and inconvenient to the user. However, since the non-secure hosts should not have critical or vital information, security maintenance can be minimal.

To provide slightly more convenient access to the Internet, another alternative for secure access is to have a firewall host which does not allow any traffic to pass through (i.e. it doesn't route traffic), but will allow both incoming and outgoing connections. Users would have accounts on this host and could access the Internet only when logged in here. For example, in Figure 1, if a user wanted to transfer a file from host A to host X, s/he would first have to transfer the file from A to the firewall host and then log into the firewall and transfer the file to host X. This solution is still not optimal in terms of user convenience, but has the advantage that security intrusions are limited to a single point of access. Unfortunately, the number of users requiring access to this host makes maintaining the security a difficult task.

FIGURE 1. Firewall Gateway Physical Connectivity



Removing the firewall host and replacing it with a router which can filter packets based on their source/destination host and port addresses can also be used to provide secure access. A reasonable filtering scheme is to allow all outbound traffic, but prohibit inbound traffic to low numbered TCP ports (i.e. less than 1024)[2]. This solution is very convenient for users who can now have Internet services directly available from their own workstations, but prohibits unwanted external access. A major problem with this design, however, is that if security on the router is compromised, all hosts on the internal network are then wide open to the Internet.

Since none of these solutions appear to be ideal, the Socks package was created to attempt to provide the best features of these methods, while keeping security problems and maintenance to a minimum. Socks

---

2. Ports less than 1024 are reserved for well-known network services (i.e. finger, ftp, telnet); ports greater than this are allocated as needed by the UNIX operating system and this is generally where outbound port numbers are obtained.

automates the process of having a firewall host which is utilized as a transient point for Internet access, making the firewall host a much more convenient security strategy, while still limiting the possibility of security intrusions to a single point of direct Internet connectivity. Although Socks does not enhance the security of the host it runs on, the simplicity and convenience of the Socks package, along with the lack of maintenance required, make it a better mechanism for securing Internet accessibility through a firewall and providing a more secure access method to the local network in general.

2.0 The Socks Package

From the point of view of a user behind the firewall host (i.e. within the local area network), there is no apparent difference between running Socks and the regular client software on a host. All connections at the application level will appear to work the same, with the hidden difference that all traffic is passing through sockd on the firewall host. This transparency is achieved through the Socks library routines which applications use in place of the normal socket library calls.

2.1 The Socks Library

The Socks library calls establish connections to sockd on the firewall and transmit information such that the daemon may perform the operation as if it was originating the request. Any data the daemon receives from the external connection will then be passed on to the original requestor (i.e. to the internal host, everything appears as usual, but to the external host, the daemon appears as the originator of the communication).

The Socks library routines are designed to propagate all network connections to the Socks daemon running on the firewall. The functions provided are designated by an "R" preceding the name of the normal C library socket calls which they are replacing (e.g. connect() becomes Rconnect()). See Table 1 for a complete list of these functions. The Socks routines take the same parameters as the original functions (with the exception of Rbind).

TABLE 1. Socks Library Routines

Function	Parameters
Rconnect	(int socket, struct sockaddr *name, int namelen)
Rbind	(int socket, struct sockaddr *name, int namelen, *struct sockaddr *remote*)
Rlisten	(int socket, int backlog)
Rgetsockname	(int socket, struct sockaddr *name, int *namelen)
Raccept	(int socket, struct sockaddr *addr, int *addrlen)

Rbind()'s additional parameter is the address of the remote host from which the connection will be established such that the daemon can refuse other, possibly hostile, connections.

2.2 The Socks Protocol

The protocol used between the Socks library routines and the daemon running on the firewall simply consists of two commands:

CONNECT <ip_address> <port number> <username>

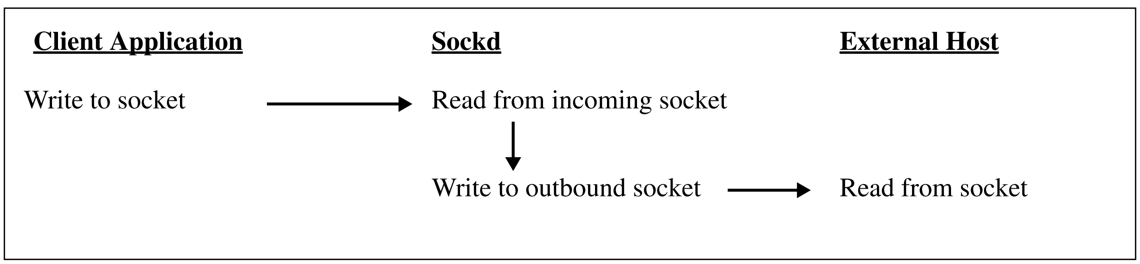
BIND <ip_address> <username>

The CONNECT command requests that the daemon establish an outbound connection to the given address and port number, while BIND requests an inbound connection expected from the given external address. The username field is a string passed from the requesting host to sockd, containing the requestor's username for the purposes of logging.

2.3 Sockd

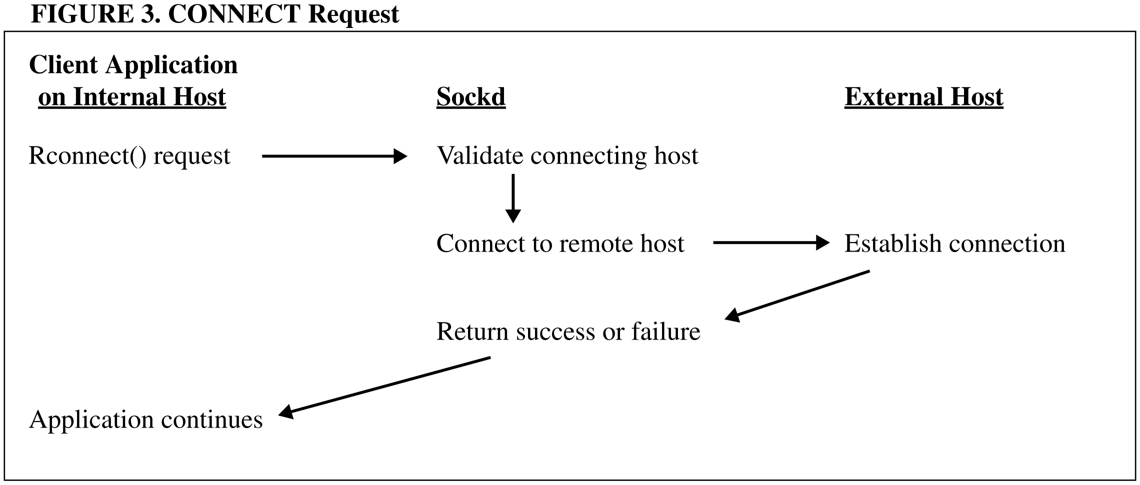
The Socks daemon (sockd) is started by inetd on a firewall host and accepts connections only from approved hosts (as determined through a configuration file, discussed in section 3.1). Applications running on these hosts may utilize the Socks library routines, presented in section 2.1, to communicate with the daemon. All attempts to establish connections are logged with both username and originating host and the daemon performs either of the actions requested through the Socks protocol: CONNECT or BIND and operates as a transient point for socket connections (see Figure 2 for an example of how a typical write() to a Socks socket would appear).

FIGURE 2. Sockd as a transient socket server



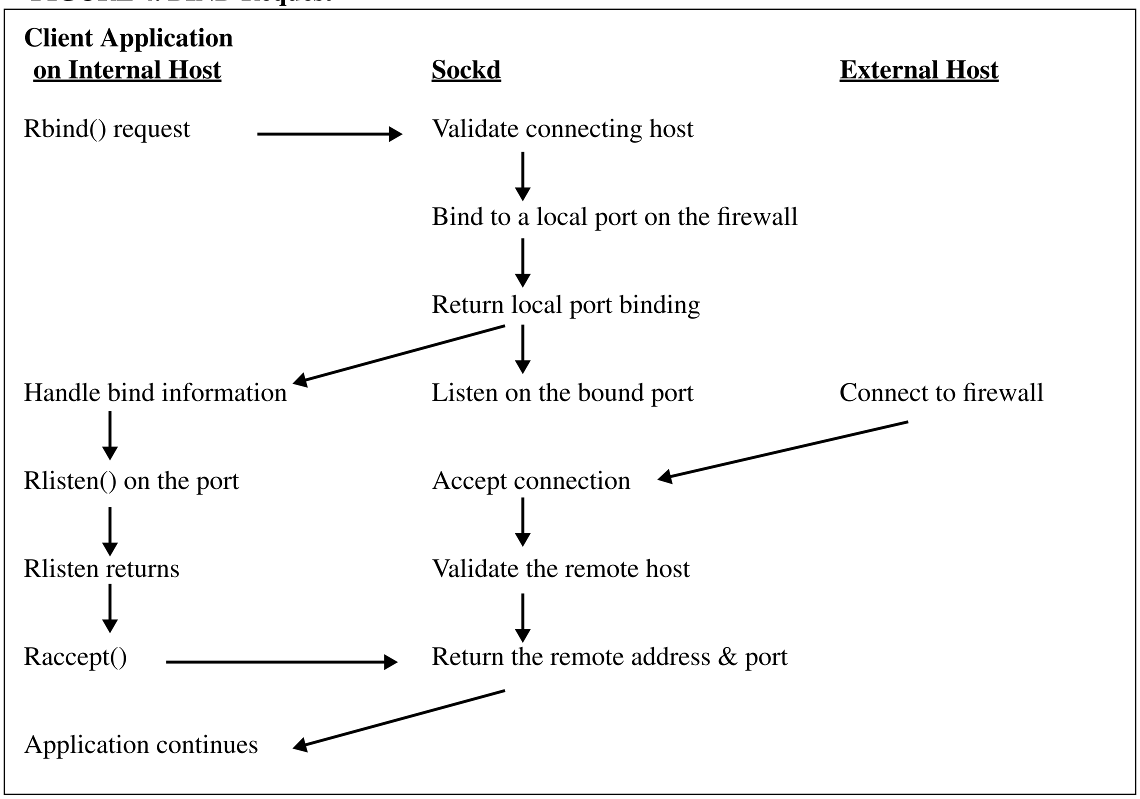
CONNECT request are originated by a call to Rconnect() on the internal host and cause the daemon to establish a connection to the remote host and return a success or fail response. At this point, the application can then read and write to the socket connection to the firewall and sockd will simply act as a bridge between the local and external socket connections. Refer to Figure 3 for an example of how the CONNECT request works.

FIGURE 3. CONNECT Request



BIND requests are slightly more complicated, but follow the same principle idea. Figure 4 shows an example of this process. The sequence of events begins when Rbind() connects to sockd which binds a new socket connection to a free port on the firewall. If successful, sockd returns the firewall port to which this connection was bound. The daemon then assumes that a bind command will be followed by listen() and accept() and performs these actions. The client can then call Rlisten(), a stub routine which always returns successfully. The next call, to Raccept(), waits for a second packet from the daemon, containing the remote host address and port from which a connection was established. This second packet can also return a failure which might be caused by either a resource failure or a connection received from a different host than specified in the BIND request. Now the connection is in a state such that all reads and writes to the socket will pass through the firewall between the internal and remote hosts.

FIGURE 4. BIND Request



3.0 Implementation

The Socks package has been implemented at MIPS Computer Systems, Inc., where there is a single host which connects to the Internet. Client applications which have been modified to work with the Socks library include ftp, telnet, finger, and whois. These applications have been renamed rftp, rtelnet, rfinger, and rwhois, respectively. This section will look at the issues involved in setting up the Socks package.

3.1 Configuration File Format

The configuration file is located on the firewall host and is used by sockd when determining whether to accept or deny requests. The file is parsed from beginning to end, with the first fully matching line returning the accessibility. The syntax of the lines in this file is as follows:

    {permit | deny} <source-host> <mask> [<dest-host> <mask> [<operator> <port>]]

Lines begin with either “permit” or “deny” following which are either 2, 4, or 6 fields, containing host address and mask pairs for source and destination, as well as a boolean operator and a service port. The host address and mask pairs are based on the syntax used by Cisco, Inc. routers and may appear backwards to those accustomed to dealing with ifconfig netmasks. The algorithm is such that the incoming host address is ANDed with the binary NOT of the mask to determine if the address matches that in the file:

<requesting address> AND (NOT <config mask>) = <config address> ?

Host addresses and services may be specified either by name or number and the boolean operators allowed are neq, eq, lt, gt, le, and ge. Access is denied to all addresses which do not match anything in the configuration file. Figure 5 shows an example of how the lines in a configuration file might appear.

FIGURE 5. A Sample Configuration File

```
#
# Deny all host to every host whois service
#
deny 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255 eq whois
#
# Let lloyd.mips.com only use finger service to sgi.com
#
permit lloyd.mips.com 0.0.0.0 sgi.com 0.0.0.0 eq finger
deny lloyd.mips.com 0.0.0.0 sgi.com 0.0.0.0
#
# Allow all hosts on the 130.62 network access to the world
#
permit 130.62.0.0 0.0.255.255
#
# Deny all hosts which do not match anything in this file
# (i.e. All hosts coming in from the Internet)
#
```

3.2 Logging

The Socks daemon records information via the UNIX syslog interface and there are three classes of messages which are logged:

- Access Denied
- Successful Connection
- Resource failure, (e.g. Out of File Descriptors, No More Processes)

The first two of these messages are the most interesting. The “Access Denied” and “Successful Connection” log entries designate where the request originated, including both host and username information, as well as the type of request (CONNECT or BIND). Information is recorded whenever a request is made of the daemon.

While logging successful BIND requests at first might appear to be useful, the more useful messages are those generated by CONNECT requests. The problem with BINDs is that since every BIND or CONNECT request creates a new daemon process, it is difficult to correlate an ftp interaction, which is what the BIND request is primarily designed to handle.

3.3 Problems with the Present Implementation

There have been no serious problems discovered in the current implementation of Socks running at MIPS Computer Systems, Inc. in the more than three years it has been in place, however there are some minor inconveniences which must be addressed: setting up the system and teaching users to use the Socks library for Internet access, and performance through the firewall host.

Setting up the Socks package is fairly simple. One configures the access file and puts the daemon in place. The more time-consuming task is modifying the client applications. This is a tedious task and it poses problems for users wishing to use the Internet access, since they also have to be fluent in the Socks package to change things. Thus, many cool tools are left by the wayside (archie, X Windows software), while their systems administrators are busy doing other things.

Performance could potentially be another problem with the Socks package arrangement, though in the years of using Socks at MIPS, no serious performance problems were ever noticed or mentioned. Had the connection been from a 10Mb/s network to 10Mb/s rather than a 10Mb/s to 1.4Mb/s network it might have been possible to notice that the intermediate gateway host provided for some lag. But since all that the daemon does after establishing the connection is read and write data in a tight loop, performance is more dependent on the speed of the network interfaces than the daemon overhead. Additionally, not allowing users accounts on the firewall host greatly increases the amount of processing power the host has for simply dealing with the data flow.

4.0 Conclusion

Although most methods of providing a secure environment require the user to make significant changes to his/her work habits, the Socks package can easily be built into familiar applications with no noticeable difference to the user. The simple configuration of the Socks daemon requires little or no maintenance and keeps users from being forced to log into the firewall host in order to utilize the available Internet resources. Although the Socks daemon does not enhance the security of the host is runs on, having a firewall host limits Internet accessibility to a single point and Socks makes this security strategy much more convenient to use. Thus, Socks provides a unique and useful solution to the problem of allowing users access to the resources of the Internet while maintaining the network integrity provided by a firewall.

References

[1] Cheswick, Bill, The Design of a Secure Internet Gateway, USENIX proceedings.

[2] Ranum, Marcus J., A Network Firewall, Digital Equipment Corporation.