SOCKS: A protocol for TCP proxy across firewalls

Ying-Da Lee
yingda@best.com or yingda@esd.sgi.com

SOCKS was originally developed by David Koblas and subsequently modified and extended by me to its current running version -- version 4. It is a protocol that relays TCP sessions at a firewall host to allow application users transparent access across the firewall. Because the protocol is independent of application protocols, it can be (and has been) used for many different services, such as telnet, ftp, finger, whois, gopher, WWW, etc. Access control can be applied at the beginning of each TCP session; thereafter the server simply relays the data between the client and the application server, incurring minimum processing overhead. Since SOCKS never has to know anything about the application protocol, it should also be easy for it to accommodate applications which use encryption to protect their traffic from nosey snoopers.

Two operations are defined: CONNECT and BIND.

1) CONNECT

The client connects to the SOCKS server and sends a CONNECT request when it wants to establish a connection to an application server. The client includes in the request packet the IP address and the port number of the destination host, and userid, in the following format.

```
                +----+----+----+----+----+----+----+----+----+----+....+----+
                | VN | CD | DSTPORT |      DSTIP        | USERID       |NULL|
                +----+----+----+----+----+----+----+----+----+----+....+----+
 # of bytes:       1    1      2              4           variable       1
```

VN is the SOCKS protocol version number and should be 4. CD is the SOCKS command code and should be 1 for CONNECT request. NULL is a byte of all zero bits.

The SOCKS server checks to see whether such a request should be granted based on any combination of source IP address, destination IP address, destination port number, the userid, and information it may obtain by consulting IDENT, cf. RFC 1413. If the request is granted, the SOCKS server makes a connection to the specified port of the destination host. A reply packet is sent to the client when this connection is established, or when the request is rejected or the operation fails.

```
                +----+----+----+----+----+----+----+----+
                | VN | CD | DSTPORT |      DSTIP        |
                +----+----+----+----+----+----+----+----+
 # of bytes:       1    1      2              4
```

VN is the version of the reply code and should be 0. CD is the result code with one of the following values:

```
        90: request granted
        91: request rejected or failed
        92: request rejected becasue SOCKS server cannot connect to
            identd on the client
        93: request rejected because the client program and identd
            report different user-ids
```

The remaining fields are ignored.

The SOCKS server closes its connection immediately after notifying
the client of a failed or rejected request. For a successful request,
the SOCKS server gets ready to relay traffic on both directions. This
enables the client to do I/O on its connection as if it were directly
connected to the application server.

2) BIND

The client connects to the SOCKS server and sends a BIND request when
it wants to prepare for an inbound connection from an application server.
This should only happen after a primary connection to the application
server has been established with a CONNECT. Typically, this is part of
the sequence of actions:

```
-bind(): obtain a socket
-getsockname(): get the IP address and port number of the socket
-listen(): ready to accept call from the application server
-use the primary connection to inform the application server of
 the IP address and the port number that it should connect to.
-accept(): accept a connection from the application server
```

The purpose of SOCKS BIND operation is to support such a sequence
but using a socket on the SOCKS server rather than on the client.

The client includes in the request packet the IP address of the
application server, the destination port used in the primary connection,
and the userid.

```
                +----+----+----+----+----+----+----+----+----+----+....+----+
                | VN | CD | DSTPORT |      DSTIP        | USERID       |NULL|
                +----+----+----+----+----+----+----+----+----+----+....+----+
 # of bytes:       1    1      2              4           variable       1
```

VN is again 4 for the SOCKS protocol version number. CD must be 2 to
indicate BIND request.

The SOCKS server uses the client information to decide whether the
request is to be granted. The reply it sends back to the client has
the same format as the reply for CONNECT request, i.e.,

```
                +----+----+----+----+----+----+----+----+
                | VN | CD | DSTPORT |      DSTIP        |
                +----+----+----+----+----+----+----+----+
 # of bytes:       1    1      2              4
```

VN is the version of the reply code and should be 0. CD is the result
code with one of the following values:

```
        90: request granted
        91: request rejected or failed
        92: request rejected becasue SOCKS server cannot connect to
            identd on the client
        93: request rejected because the client program and identd
            report different user-ids.
```

However, for a granted request (CD is 90), the DSTPORT and DSTIP fields
are meaningful. In that case, the SOCKS server obtains a socket to wait

for an incoming connection and sends the port number and the IP address of that socket to the client in DSTPORT and DSTIP, respectively. If the DSTIP in the reply is 0 (the value of constant INADDR_ANY), then the client should replace it with the IP address of the SOCKS server to which the cleint is connected. (This happens if the SOCKS server is not a multi-homed host.) In the typical scenario, these two numbers are made available to the application client prgram via the result of the subsequent getsockname() call. The application protocol must provide a way for these two pieces of information to be sent from the client to the application server so that it can initiate the connection, which connects it to the SOCKS server rather than directly to the application client as it normally would.

The SOCKS server sends a second reply packet to the client when the anticipated connection from the application server is established. The SOCKS server checks the IP address of the originating host against the value of DSTIP specified in the client's BIND request. If a mismatch is found, the CD field in the second reply is set to 91 and the SOCKS server closes both connections. If the two match, CD in the second reply is set to 90 and the SOCKS server gets ready to relay the traffic on its two connections. From then on the client does I/O on its connection to the SOCKS server as if it were directly connected to the application server.

For both CONNECT and BIND operations, the server sets a time limit (2 minutes in current CSTC implementation) for the establishment of its connection with the application server. If the connection is still not establiched when the time limit expires, the server closes its connection to the client and gives up.