Expires: April 24, 1995
<draft-ietf-aft-socks-protocol-v5-00.txt>

M. Leech
M. Ganis
Y. Lee
R. Kuris
D.
Koblas

SOCKS Protocol Version 5

Status of this Memo

Acknowledgments

This memo describes a protocol that is an evolution of the previous version of the protocol, version 4. This new protocol stems from active discussions and prototype implementations. The key contributors are: Marcus Leech: Bell-Northern Research, David Koblas: Independent Consultant, Ying-Da Lee: NEC Systems Laboratory, Lamont Jones: Hewlett-Packard, Ron Kuris: Unify Corporation, Matt Ganis: International Business Machines.

1. **Introduction**

The use of network firewalls, systems that effectively isolate an organizations internal network structure from an exterior network, such as the INTERNET is becoming increasingly popular. These firewall systems typically act as application-layer gateways between networks, usually offering controlled TELNET, FTP, and SMTP access. With the emergence of more sophisticated application layer protocols designed to facilitate global information discovery, there exists a need to provide a general framework for these protocols to transparently and securely traverse a firewall.

There exists, also, a need for strong authentication of such traversal in as fine-grained a mannner as is practical. This requirement stems from the realization that client-server relationships emerge between the networks of various organizations, and that such relationships need to be controlled and often strongly authenticated.

The protocol described here is designed to provide a framework for client-server applications in both the TCP and UDP domains to conveniently and securely use the services of a network firewall.

2. Existing practice

There currently exists a protocol, SOCKS Version 4, that provides for unsecured firewall traversal for TCP-based client-server applications, including TELNET, FTP and the popular information-discovery protocols such as HTTP, WAIS and GOPHER.

This protocol extends the SOCKS Version 4 model to include UDP, and extends the protocol header to include provisions for generalized strong authentication schemes, and extends the addressing scheme to encompass domain-name and extended IP addresses. The implementation of the SOCKS protocol typically involves the recompilation or relinking of TCP-based client applications to use the appropriate encapsulation routines in the SOCKS library.

3. Procedure for TCP-based clients

When a TCP-based client wishes to establish a connection to an object that is reachable only via a firewall (such determination is left up to the implementation), it must open a TCP connection to the appropriate SOCKS port on the SOCKS server system. The SOCKS service is conventionally located on TCP port 1080. If the connection request succeeds, the client sends a request to the server with its desired destination address, destination port, and authentication information. The SOCKS server evaluates the information, and either establishes the appropriate connection or denies it.

The SOCKS request is formed as follows:

```
+-----+------+-----+----------+----------+-----+-----+------/
|  8  |  8   |  8  |    16    |   var    |  4  |  4  |  8   /
| VER | ATYP | CMD | DST.PORT | DST.ADDR | ULN | ILN | ALEN /
|     |      |     |          |          |     |     |      /
+-----+------+-----+----------+----------+-----+-----+------/
```

```
/---------+-----------+----------+
/   var   |    var    |   var    |
/ SRC.USR | SRC.IDENT | SRC.AUTH |
/         |           |          |
/---------+-----------+----------+
```

Where:

```
  o VER       protocol version: X'05'

  o ATYP      address type of following address

                IP V4 address: X'01'
                IP V5 address: X'02'
                DOMAINNAME: X'03'
                IPNG address: X'04'

  o DST.ADDR      desired destination address

  o DST.PORT      desired destination port in network byte order

  o CMD       command: CONNECT X'01' BIND X'02'

  o ULN       length of SRC.USR field in bytes: 4 bits

  o ILN       length of SRC.IDENT field in bytes: 4 bits

  o ALEN      length of SRC.AUTH field in bytes: 8 bits

  o SRC.USR   username as known to the client-side operating system

  o SRC.IDENT identifier as known to the authentication system

  o SRC.AUTH  authenticator as known to the authentication system
```

4. Addressing

In an address field (DST.ADDR, BND.ADDR), the ATYP field specifies the type of address contained within the field. If ATYP is X'01', the address is version-4 IP address, if it is X'02', then the field specifies a version-5 IP address (that is, IP protocol version 4/5, not SOCKS protocol version 4/5). If the ATYP field is X'03', then the address field contains a DNS-style domain name, if it is X'04' then the field specifies an IPNG address.

If the ATYP field is X'01', the length of BND.ADDR is 4 bytes, if

X'02' the length is 8 bytes. If the ATYP field is X'03', then the first byte of the BND.ADDR specifies the length, in bytes, of the rest of the field.

5. Replies

The SOCKS request information is sent by the client as soon as it has established a connection to the SOCKS server. The server evaluates the request, and returns a reply formed as follows:

```
+-----+------+-----+----------+----------+
|  8  |  8   |  8  |   16     |  var     |
| VER | ATYP | REP | BND.PORT | BND.ADDR |
|     |      |     |          |          |
+-----+------+-----+----------+----------+
                ||
                \/
             +---------------+
             | 3  | 1 |  4   |
             | ET | R | CODE |
             |    |   |      |
             +----+---+------+
```

o VER   protocol version: X'05'

o ATYP   address type of following address

IP V4 address: X'01'
IP V5 address: X'02'
DOMAINNAME: X'03'
IPNG address: X'04'

o BND.ADDR   server bound address

o BND.PORT   server bound port in network byte order

o REP   Reply field:

The reply field is broken up into a 3 subfields:

ET encryption type: 3 bits

X'0' unencrypted
X'1' DES
X'2' IDEA
X'3' PRIVATE_1
X'4' PRIVATE_2
X'5' PRIVATE_3

```
                   R reply bit: always set for replies

                   CODE reason code 4 bits:

                     X'0' succeeded
                     X'1' general failure
                     X'2' bad/unknown identifier
                     X'3' connection not allowed by ruleset
                     X'4' authentication failure
                     X'5' identifier explicitly blocked
```

In a reply, the BND.ADDR and BND.PORT fields are the SOCKS server address and port number of the outbound connection for a CONNECT request, and contain the SOCKS server bind() address for a BIND request. If a reply contains a non-zero ET subfield, the server expects that there will be a bi-directional encryption of user-data on this connection using the encryption type specified in the ET subfield. It is expected that the server and client have already negotiated the appropriate key in an `out-of-band' process. It is typically the case that the same key that is used for authentication is used for encryption.

The BIND request is used in protocols which require the client to accept connections from the server. FTP is a well-known example, which uses the primary client-to-server connection for commands and status reports, but may use a server-to-client connection for transferring data on demand (e.g. LS, GET, PUT).

It is expected that the client side of an application protocol will use the BIND request only to establish secondary connections after a primary connection is established using CONNECT. Usually, then, DST.PORT and DST.ADDR in a BIND request header would be the same as those used in the primary connection, though this is not required.

In a similar fashion to CONNECT, the SOCKS server may use DST.PORT and DST.ADDR in evaluating the BIND request.

Two replies are sent from the SOCKS server to the client during a BIND operation. The first is sent after the server creates and binds a new socket. The BND.PORT field contains the port number assigned in the bind() call. The BND.ADDR field contains the associated IP address. The client will typicallly use these pieces of information to notify (via the primary or control connection) the application server of the `rendezvous point'. The second reply occurs only after the anticipated incoming connection succeeds or fails. In the second reply, only the REP field is meaningful.

When a reply (CODE value other than X'0') indicates a failure, the

SOCKS server will terminate the TCP connection shortly after sending the reply.

6. Procedure for UDP-based clients

With UDP-based clients, there is no notion of a connection, so each datagram that is to be carried by a SOCKS-UDP server must carry destination and authentication information with it. A UDP-based client must send its datagrams to the SOCKS-UDP server at UDP port 1080. Each UDP datagram carries a SOCKS request header with it:

```
+-----+------+-----+----------+----------+-----+-----+------/
|  8  |  8   |  8  |    16    |   var    |  4  |  4  |  8   /
| VER | ATYP | CMD | DST.PORT | DST.ADDR | ULN | ILN | ALEN /
|     |      |     |          |          |     |     |      /
+-----+------+-----+----------+----------+-----+-----+------/

/---------+-----------+----------+
/   var   |   var     |   var    |
/ SRC.USR | SRC.IDENT | SRC.AUTH |
/         |           |          |
/---------+-----------+----------+
```

Where:

o VER   protocol version: X'05'

o ATYP   address type of following address

IP V4 address: X'01'
IP V5 address: X'02'
DOMAINNAME: X'03'
IPNG address: X'04'

o DST.ADDR   desired destination address

o DST.PORT   desired destination port in network byte order

o CMD   command: RELAY X'03'

o ULN   length of SRC.USR field in bytes: 4 bits

o ILN   length of SRC.IDENT field in bytes: 4 bits

o ALEN   length of SRC.AUTH field in bytes: 8 bits

o SRC.USR   username as known to the client-side operating system

- o SRC.IDENT identifier as known to the authentication system

- o SRC.AUTH authenticator as known to the authentication system

When a SOCKS-UDP server decides to RELAY a UDP datagram, it does so silently, without any notification to the requesting client. Similarly, it will drop datagrams it cannot or will not RELAY. When a SOCKS-UDP server receives a reply datagram from a remote host, it will encapsulate that datagram using the standard SOCKS request header, and use the DST.ADDR and DST.PORT fields to give the originating host address and port number. When a SOCKS-UDP server receives a RELAY request from a client, it establishes a temporary association between the client address/port and a port on the SOCKS-UDP server. This temporary association is used to allow UDP reply datagrams to be correctly relayed back to the requesting UDP client. The timer related to this association is implementation dependant, but must be at least five minutes.

7. Authentication and identification information

The standard SOCKS request header includes information intended to identify the originating entity of the corresponding connection request or datagram relay request. The SRC.USR field is intended as a low-to-medium confidence mechanism for a SOCKS relay agent to provide usage auditing information only, and not as an authentication mechanism. The SRC.IDENT and SRC.AUTH fields are used to carry information that a SOCKS relay agent (server) may use to control and authenticate access to its relay services.The fundamental notion being that SRC.IDENT carries a unique identity associated with a requesting entity (typically a person) and SRC.AUTH carries some type of strong proof of that identity. In this way, the SOCKS relay agent may make decisions about access controls based on a strong notion of the identity of the entity requesting access.

In one implementation, the SRC.IDENT field carries a unique identifier that has associated with it a secret key that is shared between the relay agent and the requesting entity. The corresponding SRC.AUTH field contains time-varying information that is computed based on the shared secret key and a strong one-way hash of the time-varying data. In this implementation the shared secret key has a specific and relatively short lifetime; `out-of-band' techniques are used from time to time to assign a new secret key. In this way, the secret key acts much like a session key in Kerberos.

Another implementation may choose to use the SRC.IDENT and SRC.AUTH fields to carry information produced by so-called smart card technology to authenticate access.

Another implementation may choose to carry the operating system username in SRC.IDENT and the corresponding access password in SRC.AUTH. In this way, the authentication provided is no weaker than that provided by the FTP protocol.

8. Encrypted connections

The TCP SOCKS server may return a connection-success reply with the encryption-type (ET) field set non-zero. If this is the case, then the SOCKS server expects that all data transferred on the connection after the initial SOCKS connect request will be encrypted in a mutually agreed upon key using the algorithm specified by the SOCKS server. In order to make this work, the server expects the data stream to be encapsulated into a series of encrypted payloads as follows:

```
+------+-------+-----+----------------------------------------------+
|  8   |  128  | 16  |        var                             |     |
| PLEN | AUTHN | SSN |       DATA                             | PAD |
|      |       |     |                                        |     |
+------+-------+-----+----------------------------------------------+
```

The AUTHN field is 16-bytes long, and is expected to contain the cryptographic checksum (message digest) of the AUTHN, SSN and DATA fields prepended with the key associated with this connection. When computing this checksum, the AUTHN field is set to all X'00'. The SSN field is a unique sequence number for each encrypted payload, this is an unsigned 16-bit value transmitted in network byte order.

The receiver of an encrypted payload shall use the PLEN field, appropriately rounded for the blocksize of the encryption algorithm in use, to determine the number of bytes of encrypted data that follow. This is necessary, since the transport in use (TCP) is a stream protocol, and does not preserve I/O boundaries across a connection.

The receiver of an encrypted payload for which the received AUTHN field fails to match the computed value of AUTHN must immediately terminate the connection, and log an error to the system log. Similarly, if a received SSN doesn't increment the current received SSN, the payload is a replay, and must terminate the connection.

Since the encryption is bidirectional, the SOCKS server uses the same encapsulation technique when sending encrypted data towards the client.

9. References

[1] Koblas, D., "SOCKS", Proceedings: 1992 Usenix Security Symposium

[2] Leech, M., et al, "Socks Protocol Version 4" RFCXXXX

Authors Address

Marcus Leech, Bell-Northern Research
P.O. Box 3511, Stn. C,
Ottawa, ON
CANADA K1Y 4H7

Email: mleech@bnr.ca

Phone: (613) 763-9145