AFT Working Group   Marc VanHeyningen
draft-ietf-aft-socks-pro-v5-05.txt   Aventail Corp.
Expires six months from -->   June 27, 2000

SOCKS Protocol Version 5

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This document is a submission to the IETF Authenticated Firewall Traversal (AFT) Working Group. Comments are solicited and should be addressed to the working group mailing list (aft@socks.nec.com) or to the editor.

Abstract

This document is an update to RFC 1928, the SOCKS version 5 protocol. SOCKS is a generic proxying protocol for traversing firewalls and other trust boundaries; version 5 of the protocol adds new features such as authentication and UDP support. Changes from the RFC in this draft include formatting cleanups, authentication clarification, and fixing UDP-related problems found during implementation.

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 manner 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. The protocol is conceptually a "shim-layer" between the application layer and the transport layer, and as such does not provide network-layer gateway services, such as forwarding of ICMP messages.

2. SOCKS history

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 new protocol extends the SOCKS Version 4 model to include UDP, and extends the framework to include provisions for generalized strong authentication schemes, and extends the addressing scheme to encompass domain-name and V6 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. Connection and authentication negotiation

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 enters a negotiation for the authentication method to be used, authenticates with the chosen method, then sends a relay request. The SOCKS server evaluates the request, and either establishes the appropriate connection or denies it.

Note:

Unless otherwise noted, the decimal numbers appearing in packet-format diagrams represent the length of the corresponding field, in octets. Where a given octet must take on a specific value, the syntax X'hh' is used to denote the value of the single octet in that field. When the word 'Variable' is used, it indicates that the corresponding field has a variable length defined either by an associated (one or two octet) length field, or by a data type field.

The client connects to the server, and sends a version identifier/method selection message:

```
+----+----------+----------+
|VER | NMETHODS | METHODS  |
+----+----------+----------+
| 1  |    1     | 1 to 255 |
+----+----------+----------+
```

The VER field is set to X'05' for this version of the protocol. The NMETHODS field contains the number of method identifier octets that appear in the METHODS field.

The server selects from one of the methods given in METHODS, and sends a METHOD selection message:

```
+----+--------+
|VER | METHOD |
+----+--------+
| 1  |   1    |
+----+--------+
```

If the selected METHOD is X'FF', none of the methods listed by the client are acceptable, and the client MUST close the connection.

The values currently defined for METHOD are:

- o X'00' NO AUTHENTICATION REQUIRED
- o X'01' GSSAPI
- o X'02' USERNAME/PASSWORD
- o X'03' to X'7F' IANA ASSIGNED
- o X'80' to X'FE' RESERVED FOR PRIVATE METHODS

      o X'FF' NO ACCEPTABLE METHODS

After agreeing on the authentication method, the client and server then enter a method-specific sub-negotiation. Descriptions of the method-dependent sub-negotiations appear in separate memos.

Developers of new METHOD support for this protocol SHOULD contact IANA for a METHOD number. The ASSIGNED NUMBERS document may be referred to for a current list of METHOD numbers and their corresponding protocols.

Compliant implementations MUST support NO AUTHENTICATION REQUIRED and GSSAPI, and SHOULD support USERNAME/PASSWORD. To assure secure interoperability among multiple GSSAPI implementations, the LIPKEY[RFC 2847] mechanism MUST be supported, and mechanism negotiation using SPNEGO[RFC 2478] MUST be supported.

4. SOCKS Requests

Once the method-dependent subnegotiation has completed, the client sends the request details. If the negotiated method includes encapsulation for purposes of integrity checking and/or confidentiality, these requests MUST be encapsulated in the method-dependent encapsulation.

The SOCKS request is formed as follows:

```
        +----+-----+------+------+----------+----------+
        |VER | CMD | FLAG | ATYP | DST.ADDR | DST.PORT |
        +----+-----+------+------+----------+----------+
        | 1  |  1  |  1   |  1   | Variable |    2     |
        +----+-----+------+------+----------+----------+
```

Where:

```
      o VER    protocol version: X'05'
      o CMD
         o CONNECT          X'01'
         o BIND             X'02'
         o UDP ASSOCIATE    X'03'
         o IANA Reserved    X'04' to X'7F'
         o Private methods  X'80' to X'FF'
      o FLAG              command dependent flag (defaults to X'00')
      o ATYP              address type of following address
         o IP V4 address     X'01'
         o DOMAINNAME        X'03'
         o IP V6 address     X'04'
```

```
      o DST.ADDR          desired destination address
      o DST.PORT          desired destination port (network octet order)
```

The SOCKS server will typically evaluate the request based on source and destination addresses, and return one or more reply messages, as appropriate for the request type.

5. Addressing

In an address field (DST.ADDR, BND.ADDR), the ATYP field specifies the type of address contained within the field:

o X'01'

The address is a version-4 IP address, with a length of 4 octets.

o X'03'

The address field contains a fully-qualified domain name. The first octet of the address field contains the number of octets of name that follow, there is no terminating NUL octet.

o X'04'

The address is a version-6 IP address, with a length of 16 octets.

6. SOCKS Replies

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

```
        +----+-----+------+------+----------+----------+
        |VER | REP | FLAG | ATYP | BND.ADDR | BND.PORT |
        +----+-----+------+------+----------+----------+
        | 1  |  1  |  1   |  1   | Variable |    2     |
        +----+-----+------+------+----------+----------+
```

Where:

```
    o  VER    protocol version: X'05'
    o  REP    Reply field:
       o  X'00' succeeded
       o  X'01' general SOCKS server failure
```

```
          o  X'02' connection not allowed by ruleset
          o  X'03' Network unreachable
          o  X'04' Host unreachable
          o  X'05' Connection refused
          o  X'06' TTL expired
          o  X'07' Command not supported
          o  X'08' Address type not supported
          o  X'09' Invalid address
          o  X'0A' to X'FF' unassigned
       o  FLAG   command dependent flag
       o  ATYP   address type of following address
          o  IP V4 address: X'01'
          o  DOMAINNAME:    X'03'
          o  IP V6 address: X'04'
       o  BND.ADDR server bound address
       o  BND.PORT server bound port (network octet order)
```

If the chosen method includes encapsulation for purposes of authentication, integrity and/or confidentiality, the replies are encapsulated in the method-dependent encapsulation.

When a reply indicates a failure (REP value other than X'00',) the SOCKS server MUST terminate the TCP connection shortly after sending the reply. This must be no more than 10 seconds after detecting the condition that caused a failure.

If the reply code indicates a success, the client may now start passing data. If the selected authentication method supports encapsulation for the purposes of integrity, authentication and/or confidentiality, the data are encapsulated using the method-dependent encapsulation. Similarly, when data arrives at the SOCKS server for the client, the server MUST encapsulate the data as appropriate for the authentication method in use.

7. TCP Procedure

7.1. CONNECT

In the reply to a CONNECT, BND.PORT contains the port number that the server assigned to connect to the target host, and BND.ADDR contains the associated IP address. The supplied BND.ADDR is often different from the IP address that the client uses to reach the SOCKS server, since such servers are often multi-homed. It is expected that the SOCKS server will use DST.ADDR and DST.PORT, and the client-side source address and port in evaluating the CONNECT request.

7.2. BIND

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. DST.ADDR contains the address of the primary connection's destination. DST.PORT contains the requested port (or 0 for a random, unused port). It is expected that a SOCKS server will use DST.ADDR and DST.PORT 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 that the SOCKS server assigned to listen for an incoming connection. The BND.ADDR field contains the associated IP address. The client will typically use these pieces of information to notify (via the primary or control connection) the application server of the rendezvous address. The second reply occurs only after the anticipated incoming connection succeeds or fails.

In the second reply, the BND.PORT and BND.ADDR fields contain the address and port number of the connecting host.

Note: out-of-band data

As with other TCP application data, out of band data is normally proxied to the SOCKS server as out of band data; note that implementations may be limited to handling a single byte of such data at a time. Authentication methods which define some content encapsulation SHOULD define a method-specific mechanism for proxying out of band data.

8. UDP procedure

8.1. UDP ASSOCIATE requests

The UDP ASSOCIATE request is used to establish an association within the UDP relay process to handle UDP datagrams. The DST.ADDR and DST.PORT fields contain the address and port that the client expects to use to send UDP datagrams on for the association. The server MAY use this information to limit access to the association. If the client is not in possesion of the information at the time of the UDP ASSOCIATE, the client MUST use address type X'01' with a port number and address of all zeros.

A UDP association terminates when the TCP connection that the UDP ASSOCIATE request arrived on terminates.

Flag bits in the request and reply are defined as follows:

```
o INTERFACE REQUEST X'01'
o USECLIENTSPORT    X'04'
```

If the USECLIENTSPORT bit is set in the flag field of the request, the server SHOULD interact with the application server using the same port the client used in the request, and set the USECLIENTSPORT bit in the flag field of the reply to acknowledge having done so. If no port number was specified in the UDP ASSOCIATE request, this flag is meaningless and MUST not be used.

If the INTERFACE REQUEST bit is set in the flag field of the request, the server may indicate its support for this extension by setting this bit in the reply. If both client and server support this feature, the client SHOULD send INTERFACE DATA subcommands, described below, during the UDP association.

In the reply to a UDP ASSOCIATE request, the BND.PORT and BND.ADDR fields indicate the port number/address where the client MUST send UDP request messages to be relayed.

8.2. UDP Control Channel

A UDP association terminates when the TCP connection that the UDP ASSOCIATE request arrived on terminates. If the flag negotiation indicated mutual support for it, the client SHOULD send INTERFACE DATA subcommands to learn the external address information for the UDP assocaiation with respect to a particular destination. The server, in turn, MAY use this information to limit access to the association to those destination addresses for which it has received INTERFACE DATA queries; multiple INTERFACE DATA commands are permitted, and have a cumulative effect.

Such requests are formatted as follows:

```
+----+-----+------+------+----------+------+
|RSV | SUB | FLAG | ATYP | ADDR     | PORT |
+----+-----+------+------+----------+------+
| 1  |  1  |  1   |   1  | Variable |  2   |
+----+-----+------+------+----------+------+
```

The fields in the CONTROL CHANNEL packet are:

```
o  RSV  Reserved X'00'
```

```
      o  SUB  Subcommand
         o  INTERFACE DATA: X'01'
      o  FLAG  subcommand dependent flag (normally X'00')
      o  ATYP  address type of following addresses:
         o  IP V4 address: X'01'
         o  DOMAINNAME:    X'03'
         o  IP V6 address: X'04'
      o  ADDR  destination address information
      o  PORT  destination port information (network octet order)
```

Replies to INTERFACE DATA commands are structured the same way as ordinary SOCKS replies, as per section 6.

8.3. UDP packet structure

A UDP-based client MUST send its datagrams to the UDP relay server at the UDP port indicated by BND.PORT in the reply to the UDP ASSOCIATE request. If the selected authentication method provides encapsulation for the purposes of authenticity, integrity, and/or confidentiality, the datagram MUST be encapsulated using the appropriate encapsulation. Each UDP datagram carries a UDP request header with it:

```
      +------+------+------+----------+----------+----------+
      | FLAG | FRAG | ATYP | DST.ADDR | DST.PORT |   DATA   |
      +------+------+------+----------+----------+----------+
      |  2   |  1   |  1   | Variable |    2     | Variable |
      +------+------+------+----------+----------+----------+
```

The fields in the UDP request header are:

```
        o  FLAG     Reserved (currently X'0000')
        o  FRAG     Current fragment number
        o  ATYP     address type of following addresses:
           o  IP V4 address: X'01'
           o  DOMAINNAME:    X'03'
           o  IP V6 address: X'04'
        o  DST.ADDR desired destination address
        o  DST.PORT desired destination port (network octet order)
        o  DATA     user payload data
```

FRAG is currently unused, and reserved for future work to deal with fragmentation; it must be set to X'00'.

When a UDP relay 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 UDP relay server receives a reply datagram from a remote host, it MUST encapsulate that datagram using the above UDP request header, and any authentication-method-dependent encapsulation.

The UDP relay server MUST acquire from the SOCKS server the expected IP address of the client that will send datagrams to the BND.PORT given in the reply to UDP ASSOCIATE. It MUST drop any datagrams arriving from any source IP address other than the one recorded for the particular association.

The programming interface for a SOCKS-aware UDP MUST report an available buffer space for UDP datagrams that is smaller than the actual space provided by the operating system:

- if ATYP is X'01' - 10+method_dependent octets smaller
- if ATYP is X'03' - 262+method_dependent octets smaller
- if ATYP is X'04' - 20+method_dependent octets smaller

9. Security Considerations

This document describes a protocol for the application-layer traversal of IP network firewalls. The security of such traversal is highly dependent on the particular authentication and encapsulation methods provided in a particular implementation, and selected during negotiation between SOCKS client and SOCKS server.

Careful consideration should be given by the administrator to the selection of authentication methods.

10. Acknowledgments

This memo describes a protocol that is an evolution of the previous version of the protocol, version 4[SOCKS]. 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 Company
- Ron Kuris: Unify Corporation
- Matt Ganis: International Business Machines
- David Blob: NEC USA
- Wei Lu: NEC USA.
- William Perry: Aventail

o Dave Chouinard: Intel

11. References

[GSSAPI]   Margrave, D., "GSS-API Authentication Method for SOCKS Version 5," work in progress.

[RFC 1928] Leech, M., Ganis, M., Lee, Y., Kuris, R. Koblas, D., & Jones, L., "SOCKS Protocol V5," RFC 1928, April 1996.

[RFC 1929] Leech, M., "Username/Password Authentication for SOCKS V5," RFC 1929, March 1996.

[RFC 1961] McMahon, P., "GSS-API Authentication Method for SOCKS Version 5," RFC 1961, June 1996.

[RFC 2478] Baize, E. and Pinkas, D., "The Simple and Protected GSS-API Negotiation Mechanism," RFC 2478, December 1998.

[RFC 2847] Eisler, M., "LIPKEY - A Low Infrastructure Public Key Mechanism Using SPKM," RFC 2847, June 2000.

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

Author's Address

Marc VanHeyningen
Aventail Corporation
808 Howell Streeet; Suite 200
Seattle, WA 98101 USA

Phone: +1 (206) 215-1111
Email: marcvh@aventail.com