Network Working Group
Internet-Draft
Intended status: Historic
Expires: 8 May 2026

D. J. Vance
Independent

4 November 2025

SOCKS Protocol Version 4 Specification
draft-vance-socks-v4-01

Abstract

This document is published as a historical record of the SOCKS 4 protocol. The original spec does not have an Abstract, so the Abstract below is added afterwards.

This document describes SOCKS version 4, a protocol designed to facilitate TCP proxy services across a network firewall. SOCKS operates at the session layer, providing application users with transparent access to network services on the other side of the firewall. It is application-protocol independent, allowing it to support a wide range of services, including those utilizing encryption, while maintaining minimum processing overhead by simply relaying data after initial access control checks. The protocol defines two primary operations: CONNECT for establishing outbound connections to an application server, and BIND for preparing for and accepting inbound connections initiated by an application server.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/4socks/socks4.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on 8 May 2026.

Table of Contents

1. Introduction

The SOCKS protocol, Version 4 (SOCKSv4), SHALL be used to relay TCP sessions between an application client and an an application server via a SOCKS server, often positioned at a firewall host. The protocol MUST provide transparent access across the firewall for application users.

The protocol MUST be application-protocol independent, allowing it to be used for various services, including, but not limited to, telnet, ftp, finger, whois, gopher, and WWW.

The SOCKS server MUST apply access control mechanisms at the beginning of each TCP session. Following successful establishment, the SOCKS server MUST simply relay data between the client and the application server, incurring minimum processing overhead. The protocol inherently supports applications utilizing encryption, as the SOCKS server is not required to interpret the application protocol's payload.

Two primary operations are defined: CONNECT and BIND.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification uses the following terms:

* Client (Application Client): The program requesting a connection to an application server through the SOCKS server.

* SOCKS Server: The host, typically at a firewall, that intermediates the connection between the Client and the Application Server.

* Application Server: The host to which the Client ultimately wishes to connect (e.g., a Telnet daemon, an HTTP server).

* TCP Session: A connection established using the Transmission Control Protocol (TCP). SOCKSv4 only supports TCP sessions.

* DSTIP (Destination IP): The IP address of the Application Server, as specified in the SOCKS request.

* DSTPORT (Destination Port): The port number of the Application Server, as specified in the SOCKS request.

* USERID: A variable-length, NULL-terminated string identifying the client's user on the local system.

* NULL: A byte of all zero bits, used to terminate the USERID field.

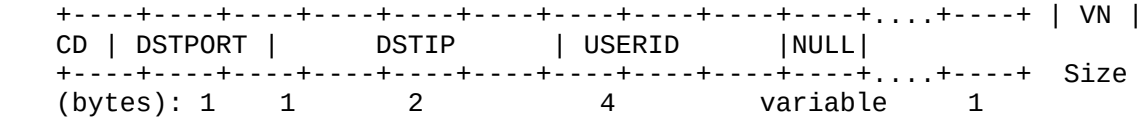
* IDENT: A protocol (as described in RFC 1413) used by the SOCKS server to verify the user identity of the client.

3. CONNECT Operation

The client MUST initiate a CONNECT request when it desires to establish an outbound TCP connection to an application server.

3.1. CONNECT Request Packet Format

The client MUST send a request packet with the following structure:



* VN (Version Number): MUST be 4, representing the SOCKS protocol version.

* CD (Command Code): MUST be 1, indicating a CONNECT request.

* DSTPORT (Destination Port): The port number of the application server (network byte order).

* DSTIP (Destination IP): The IP address of the application server (network byte order).

* USERID (User Identifier): A string of characters representing the client's user ID.

* NULL: A single byte with a value of all zero bits, terminating the USERID field.

3.2. CONNECT Processing and Reply

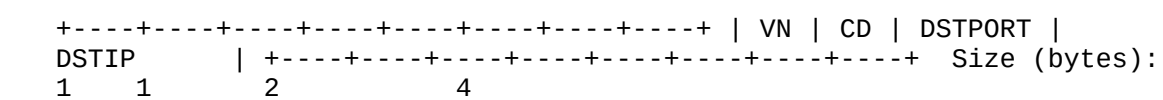
The SOCKS server MUST determine whether to grant the request based on criteria such as the source IP address, DSTIP, DSTPORT, USERID, and information obtained via IDENT (cf. RFC 1413).

If the request is granted, the SOCKS server MUST attempt to establish a TCP connection to the specified DSTPORT on the DSTIP.

A reply packet MUST be sent to the client upon the establishment of the connection, rejection of the request, or operational failure.

3.3. CONNECT Reply Packet Format

The SOCKS server MUST send a reply packet with the following structure:



+----+----+----+----+----+----+----+----+ | VN | CD | DSTPORT | DSTIP | +----+----+----+----+----+----+----+----+ Size (bytes): 1 1 2 4

- VN: MUST be 0, representing the reply version code. - CD (Result Code): The SOCKS server MUST use one of the following values: - 90: Request granted. - 91: Request rejected or failed. - 92: Request rejected due to inability to connect to identd on the client. - 93: Request rejected because the client program and identd report different user-IDs. - DSTPORT and DSTIP: These fields MUST be ignored by the client in a CONNECT reply.

If the request is rejected or failed (CD != 90), the SOCKS server MUST close its connection to the client immediately after sending the reply.

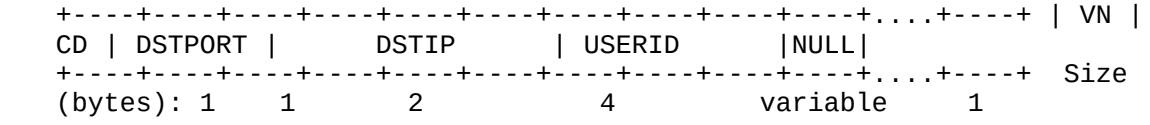
If the request is successful (CD = 90), the SOCKS server MUST immediately begin relaying traffic in both directions between the client connection and the established application server connection. The client MUST then treat its connection to the SOCKS server as if it were a direct connection to the application server.

4. BIND Operation

The client MUST initiate a BIND request when it requires the SOCKS server to prepare for an inbound connection from an application server. This operation is typically used for protocols that involve a secondary data connection originating from the server (e.g., FTP’s active mode). A BIND request SHOULD only be sent after a primary connection to the application server has been successfully established using a CONNECT request.

4.1. BIND Request Packet Format

The client MUST send a request packet identical in format to the CONNECT request:



- VN: MUST be 4. - CD: MUST be 2, indicating a BIND request. - DSTPORT: The port number of the primary connection to the application server. - DSTIP: The IP address of the application server. - USERID and NULL: As defined for the CONNECT request.

4.2. BIND First Reply (Socket Assignment)

The SOCKS server MUST first decide whether to grant the BIND request. The reply format MUST be the same as the CONNECT reply format.

If the request is rejected (CD != 90), the SOCKS server MUST close its connection to the client immediately.

If the request is granted (CD = 90):

1. The SOCKS server MUST obtain a local socket and begin listening for an incoming connection. 2. The SOCKS server MUST send a first reply packet where the DSTPORT and DSTIP fields are meaningful: - DSTPORT MUST contain the port number of the newly listening socket (network byte order). - DSTIP MUST contain the IP address of the SOCKS server’s listening interface (network byte order). 3. If the SOCKS server returns a DSTIP of 0 (the value of constant ’INADDR_ANY’), the client MUST replace this value with the IP address of the SOCKS server to which the client is currently connected. 4. The client MUST use this IP address and port to inform the application server via the primary connection, enabling the application server to initiate the anticipated inbound connection to the SOCKS server.

4.3. BIND Second Reply (Connection Established)

The SOCKS server MUST send a second reply packet to the client once the anticipated inbound connection from the application server is established. The reply format MUST be the same as the first reply.

The SOCKS server MUST check the IP address of the newly connected application server host against the DSTIP value specified in the client’s original BIND request.

- If the IP addresses match: The CD field in the second reply MUST be set to 90. The SOCKS server MUST then prepare to relay traffic between the client connection and the new application server connection. - If a mismatch is found: The CD field in the second reply MUST be set to 91. The SOCKS server MUST immediately close both the client connection and the connection from the application server.

Upon a successful second reply, the client MUST perform I/O on its connection to the SOCKS server as if it were directly connected to the application server.

5. Timeout Mechanism

For both CONNECT and BIND operations, the SOCKS server MUST employ a time limit for the establishment of its connection with the application server (e.g., 2 minutes). If the connection is not established before the time limit expires, the SOCKS server MUST close its connection to the client and abort the operation.

6. Security Considerations

The SOCKS Version 4 (SOCKSv4) protocol, designed for TCP proxy traversal of network firewalls, operates exclusively at the session layer and *inherently lacks robust security mechanisms*. Its deployment and operational policy must be rigorously evaluated against the deficiencies outlined herein.

6.1. Authentication and Authorization Deficiencies

6.1.1. Weak Client Identification Mechanism

The SOCKSv4 request format incorporates a *USERID* field. This field is designated for rudimentary client identification, typically intended for conjunction with the *IDENT protocol* (specified in [RFC 1413]).

* *Ident Protocol Vulnerability:* Reliance on IDENT constitutes a *significant security risk*. The IDENT protocol operates via an untrusted daemon resident on the client host, rendering the identification process susceptible to trivial *spoofing or malicious disabling*.

* *Absence of Strong Authentication:* SOCKSv4 *lacks integrated provisions* for strong client-to-server or server-to-client authentication. This includes the absence of any mechanism for verifying user credentials, such as passwords, or employing cryptographic challenge-response methods.

6.1.2. Policy-Dependent Authorization

Access control (authorization) for SOCKSv4 services is *exclusively managed* by the local configuration and security policy of the SOCKS server implementation. A failure in the server's configuration or a weakness in its policy can directly result in *unauthorized network access* across the protective boundary of the firewall.

6.2. Data Integrity and Transport Limitations

6.2.1. Absence of Confidentiality (Plaintext Relay)

SOCKSv4 functions as a session layer relay and *does not incorporate any encryption* capabilities for the application data stream. All application traffic traversing the SOCKS proxy is forwarded in *plaintext*. This inherent vulnerability exposes all transmitted data to *passive network eavesdropping* and interception.

6.2.2. Protocol Scope Restriction

The SOCKSv4 protocol is *strictly confined* to the proxying of *Transmission Control Protocol (TCP)* connections. It provides *no native support* for the relay of *User Datagram Protocol (UDP)* traffic or other protocols operating at the IP layer.

6.3. Vulnerabilities Associated with the BIND Operation

The *BIND* command, utilized to establish a socket for an anticipated inbound connection (a callback) from an application server, introduces distinct security challenges.

6.3.1. Source Address Verification Bypass

The SOCKS server attempts a rudimentary security check during the BIND operation by comparing the source IP address of the incoming connection with the target address (DSTIP) specified in the client's request.

* *IP Address Spoofing Risk:* A malicious actor could potentially *forge the source IP address* of the inbound connection, thereby bypassing this basic server check and facilitating the establishment of an *unauthorized session*.

* *NAT/PAT Incompatibility:* In network topologies employing *Network Address Translation (NAT)* or *Port Address Translation (PAT)*, the source IP address is structurally altered. This modification renders the BIND source address verification mechanism *unreliable, ineffectual, or operationally complex* to maintain.

6.4. Denial of Service (DoS) Vector

6.4.1. Resource Exhaustion Potential

Each successful SOCKS connection consumes finite server resources, including active sockets, allocated memory, and network bandwidth. A malicious client can exploit this by initiating a *large volume of connection attempts*—particularly through the resource-intensive *BIND operation*—to rapidly exhaust the SOCKS server's capacity. This constitutes a direct vector for a *Denial of Service* attack against legitimate users.

6.4.2. Inadequate Mitigations

Although the protocol specifies a basic connection establishment *timeout mechanism (2 minutes)*, this measure is insufficient in scope and rigor to fully mitigate the risks associated with sophisticated DoS attacks.

6.5. Recommended Mitigation and Deployment Practices

Given the security deficiencies of SOCKSv4, deployment should be guided by the following principles:

1. Strict Operational Environment: SOCKSv4 is only recommended for use in environments designated as highly trusted and subject to stringent local policy control*.

2. Layered Security via Encrypted Tunnels: Where SOCKSv4 must transport sensitive application traffic, the protocol must be encapsulated within an existing secure transport layer, such as a Transport Layer Security (TLS/SSL) or IPsec tunnel, to establish confidentiality and integrity.

3. Protocol Migration: Operators should actively plan for the migration to or substitution with a more secure protocol version, specifically SOCKS Version 5 ([RFC 1928]), which incorporates native, robust authentication methods.

7. IANA Considerations

This document describes the SOCKS Version 4 protocol, which is presented as a historical record. This protocol does not define any new protocol fields, codes, or registries that require assignment by the Internet Assigned Numbers Authority (IANA).

The existing values used within the protocol are summarized below:

7.1. SOCKS Protocol Version Number (VN)

* The SOCKS protocol version number VN in requests is *4 (0x04)*.

* The SOCKS protocol version number VN in replies is *0 (0x00)*.

7.2. SOCKS Command Code (CD)

The SOCKS command code CD in requests defines two values: * *1 (0x01):* CONNECT * *2 (0x02):* BIND

7.3. SOCKS Reply Code (CD)

The SOCKS reply code CD in replies defines four values: * *90 (0x5A):* Request granted * *91 (0x5B):* Request rejected or failed * *92 (0x5C):* Request rejected because SOCKS server cannot connect to identd on the client * *93 (0x5D):* Request rejected because the client program and identd report different user-ids

7.4. Port Number

The SOCKS protocol is conventionally known to use *TCP port 1080* for its service. This port number has already been registered in the *IANA Service Name and Transport Protocol Port Number Registry* for the socks service.

8. References

8.1. Normative References

[RFC1413] St. Johns, M., "Identification Protocol", RFC 1413, DOI 10.17487/RFC1413, February 1993, <https://www.rfc-editor.org/rfc/rfc1413>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

8.2. Informative References

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <https://www.rfc-editor.org/rfc/rfc1928>.

[RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, DOI 10.17487/RFC1929, March 1996, <https://www.rfc-editor.org/rfc/rfc1929>.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/rfc/rfc791>.

[RFC793] Postel, J., "Transmission Control Protocol", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/rfc/rfc793>.

Original Auther

Ying-Da Lee
Principal Member Technical Staff
NEC Systems Laboratory, CSTC
ylee@syl.dl.nec.com

Author’s Address

Daniel James Vance
Independent
Email: djvanc@outlook.com