Network Working Group
Internet-Draft
Intended status: Historic
Expires: 19 May 2026

D. J. Vance
Independent

15 November 2025

SOCKS Protocol Version 4 Specification
draft-vance-socks-v4-04

Abstract

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

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 (World Wide Web).

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:

Field	Description	Size (bytes)
VN	Version Number	1
CD	Command Code	1
DSTPORT	Destination Port	2
DSTIP	Destination IP Address	4
USERID	User ID	variable
NULL	Null Terminator	1

Table 1: CONNECT Request Packet Format

* 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.

When the DSTIP field is 0.0.0.1, which the protocol SOCKSv4a (See Appendix A.1) uses for a client wishes to connect using a domain name instead of an IP address, SOCKSv4 implementations MUST treat the the DSTIP field 0.0.0.1 as a normal DSTIP value and treat the following messages as the specification.

3.3. CONNECT Reply Packet Format

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

Field	Description	Size (bytes)
VN	Version Number	1
CD	Command Code	1
DSTPORT	Destination Port	2
DSTIP	Destination IP Address	4

Table 2: CONNECT Reply Packet Format

* VN: MUST be 0, representing the reply version code.

* CD (Result Code): The SOCKS server MUST use one of the following values:

Reply Code	Description
90	Request granted (Connection successful).
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.

Table 3: Result Codes

* 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:

Field	Description	Size (bytes)
VN	Version Number (must be 4)	1
CD	Command Code (1 for CONNECT, 2 for BIND)	1
DSTPORT	Destination Port (Network Byte Order)	2
DSTIP	Destination IP Address	4
USERID	User ID (String of Octets)	variable
NULL	Null Terminator (0x00)	1

Table 4: BIND Request Packet Format

* 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

See Appendix B.

7. IANA Considerations

No IANA actions required.

See Appendix C for the existing values used within the protocol.

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>.

[SOCKS4] Lee, Y.-D., "SOCKS: A protocol for TCP proxy across firewalls", n.d..

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>.

[RFC3365] Schiller, J., "Strong Security Requirements for Internet Engineering Task Force Standard Protocols", BCP 61, RFC 3365, DOI 10.17487/RFC3365, August 2002, <https://www.rfc-editor.org/rfc/rfc3365>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/rfc/rfc3552>.

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

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/rfc/rfc9293>.

Appendix A. Common Operational Extensions

The content of this appendix is Informative, not Normative. It describes extensions and interpretations of the SOCKSv4 protocol that have been widely adopted in practical deployments and client implementations to enhance functionality and compatibility.

A.1. SOCKS Protocol Version 4A

The SOCKSv4 protocol originally required the client to resolve the target domain name before sending the request. As this is impractical in many environments, the SOCKSv4a protocol was widely adopted to allow the SOCKS server to perform domain name resolution.

SOCKSv4a, though share a same version number with SOCKSv4, is treated as a complete independent protocol here. The specification will be published elsewhere. The content below is just a simple summary of SOCKSv4a, and it should never be treated as a Normative standard.

Clients using this protocol must follow these rules:

A.1.1. SOCKSv4a Request Format

When a client wishes to connect using a domain name instead of an IP address, the request format follows the CONNECT/BIND format, but with modifications to DSTIP and the end of the request:

Field	Description	Size (bytes)	SOCKSv4a Usage
VN	Version Number (4)	1	Unchanged.
CD	Command Code (1 or 2)	1	Unchanged.
DSTPORT	Destination Port	2	Unchanged.
DSTIP	Destination IP Address	4	MUST be set to 0.0.0.1 (0x00000001).
USERID	User ID	variable	Unchanged.
NULL	Null Terminator (0x00)	1	Terminates USERID.
DOMAIN	Target Domain Name	variable	New field: Null-terminated string.
NULL	Final Null Terminator	1	New field: Terminates DOMAIN.

Table 5: SOCKSv4a Request Format

A SOCKSv4a client, when sending a request, must append the target domain name string after the NULL terminator of USERID, and terminate the entire request with a second NULL byte.

A.1.2. SOCKSv4a Server Processing

When a SOCKSv4a server receives a request where the DSTIP field is 0.0.0.1, it MUST perform the following actions:

1. Treat 0.0.0.1 as a special signal and MUST NOT attempt to connect to this IP address.

2. Start reading data after the USERID’s NULL terminator, interpreting it as the target domain name string (DOMAIN), until the next NULL terminator is encountered.

3. The server MUST attempt to resolve this domain name.

4. If resolution is successful, the server attempts to connect to the obtained IP address. If the connection succeeds, it replies 90. If the connection fails, it replies 91.

5. If resolution fails, the server MUST reply 91 and close the connection.

A.2. Use of DSTIP/DSTPORT in BIND Requests for Access Control

Although DSTIP and DSTPORT in the BIND request (Section 4.1) are intended to identify the application server, many SOCKS server and firewall implementations use them as an Access Control List (ACL) for the inbound connection.

* DSTIP as Source Address Restriction: The server strictly requires the IP address of the inbound connection to MUST match the DSTIP specified in the BIND request.

* DSTPORT as Source Port Restriction (less common): Some implementations may attempt to verify that the source port of the inbound connection matches the DSTPORT in the BIND request. Since the source port of an application server is usually randomly allocated by the operating system, this usage is generally considered unreliable or misleading and is ignored in most implementations.

When initiating a BIND request, a client SHOULD ensure that DSTIP is the address of the application server it expects to receive the connection from, to improve compatibility.

A.3. Explanation of Timeout Mechanism

As mandated by Section 5, the SOCKS server MUST employ time limits. In common implementations, timeouts usually trigger under the following circumstances:

* CONNECT Timeout: The server is unable to establish a connection with DSTIP:DSTPORT within the specified time.

* Timeout after the first BIND reply: After the SOCKS server successfully binds the listening socket (sent the first 90 reply), but fails to receive an inbound connection from the application server within the specified time.

When a timeout occurs, the server MUST immediately close the connection with the client and abort all pending network operations.

Appendix B. Security Analysis

The SOCKS Version 4 (SOCKSv4) protocol, designed exclusively for TCP proxy traversal across network firewalls, is fundamentally weak from a security perspective as it operates solely at the session layer and lacks intrinsic security mechanisms. Any deployment of SOCKSv4 must be critically assessed against its inherent deficiencies.

B.1. Authentication and Authorization Deficiencies

SOCKSv4's client identification relies on the USERID field, often intended for use with the IDENT protocol (specified in RFC 1413). This reliance constitutes a major security risk because the IDENT protocol depends on an untrusted daemon on the client host, making the identification process susceptible to trivial spoofing or malicious disabling. Crucially, SOCKSv4 entirely lacks integrated provisions for strong client-to-server or server-to-client authentication, offering no mechanisms for verifying user credentials, passwords, or employing cryptographic challenge-response methods. Consequently, access control (authorization) is managed exclusively by the SOCKS server's local configuration and security policy. A failure in the server's policy or configuration directly risks granting unauthorized network access across the protective boundary of the firewall.

B.2. Data Integrity and Transport Limitations

SOCKSv4 does not incorporate any encryption capabilities for the application data stream. As a session layer relay, it forwards all application traffic, including sensitive data, in plaintext. This inherent vulnerability exposes all transmitted data to passive network eavesdropping and interception, resulting in a total absence of confidentiality. Furthermore, the protocol' s operational scope is strictly confined to proxying Transmission Control Protocol (TCP) connections. It provides no native support for the relay of User Datagram Protocol (UDP) traffic or other IP-layer protocols, limiting its utility and scope of protection.

B.3. Vulnerabilities Associated with the BIND Operation

The BIND command, used for establishing a socket for an anticipated inbound connection (a callback) from an application server, introduces distinct security challenges. The SOCKS server attempts a rudimentary security check by comparing the source IP address of the incoming connection with the target address (DSTIP) specified in the client's request. However, a malicious actor can easily forge the source IP address of the inbound connection, potentially bypassing this basic server check and facilitating an unauthorized session.

Moreover, in network topologies employing Network Address Translation (NAT) or Port Address Translation (PAT), the source IP address is structurally altered, rendering the BIND source address verification mechanism unreliable, ineffectual, or operationally complex to maintain.

B.4. Denial of Service (DoS) Vector

Every successful SOCKS connection consumes finite server resources, including active sockets, allocated memory, and network bandwidth. A direct vector for a Denial of Service attack exists where a malicious client can exploit this resource consumption by initiating a large volume of connection attempts, particularly through the resource-intensive BIND operation, to rapidly exhaust the SOCKS server's capacity. Although the protocol specifies a basic connection establishment timeout mechanism (2 minutes), this measure is entirely insufficient in scope and rigor to fully mitigate the risks associated with sophisticated DoS attacks.

B.5. Recommended Mitigation and Deployment Practices

Given SOCKSv4's security deficiencies, its deployment should be strictly limited to environments designated as highly trusted and subject to stringent local policy control. 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 the essential confidentiality and integrity mechanisms that SOCKSv4 lacks. Operators should actively plan for migration to SOCKS Version 5 (RFC 1928), which incorporates native, robust authentication methods.

Appendix C. Existing Values

The existing values used within the protocol are summarized below:

C.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).

C.2. SOCKS Command Code (CD)

The SOCKS command code CD in requests defines two values:

* 1 (0x01): CONNECT

* 2 (0x02): BIND

C.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

C.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.

Original Auther

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

David Koblas
Netskope

We sincerely apologize that, due to the document’s long history and the passage of time, many early contributors may not have been formally included in this list. We extend our deepest gratitude to all who have contributed to this work. If you believe your name should be added to the acknowledgments, please contact the draft maintainers.

Author’s Address

Daniel James Vance
Independent
Email: djvanc@outlook.com