Network Working Group
Internet-Draft
Intended status: Historic
Expires: 5 November 2026

D. J. Vance
Independent

4 May 2026

SOCKS Protocol Version 4
draft-vance-socks-v4-09

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

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 5 November 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 (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 [RFC1413]) 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. [RFC1413]).

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:

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

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):

* The SOCKS server MUST obtain a local socket and begin listening for an incoming connection.

* The SOCKS server MUST send a first reply packet in which the DSTPORT and DSTIP fields are meaningful: DSTPORT MUST contain, in network byte order, the port number of the newly listening socket, and DSTIP MUST contain, in network byte order, the IP address of the SOCKS server's listening interface.

* 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.

* 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

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 C.

7. IANA Considerations

No IANA actions required.

See Appendix D 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>.

8.2. Informative References

[I-D.vance-socks-v4a] Vance, D. J., "SOCKS Protocol Version 4A", Work in Progress, Internet-Draft, draft-vance-socks-v4a-01, 15 February 2026, <https://datatracker.ietf.org/doc/html/draft-vance-socks-v4a-01>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/rfc/rfc1918>.

[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>.

[RFC1948] Bellovin, S., "Defending Against Sequence Number Attacks", RFC 1948, DOI 10.17487/RFC1948, May 1996, <https://www.rfc-editor.org/rfc/rfc1948>.

[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>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/rfc/rfc4301>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <https://www.rfc-editor.org/rfc/rfc4732>.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <https://www.rfc-editor.org/rfc/rfc4953>.

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[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. Relationship with SOCKS 4A

The relationship between the base SOCKSv4 protocol and the variant commonly known as SOCKS 4A ([I-D.vance-socks-v4a]) is frequently misunderstood as a simple backward-compatible extension. In strict architectural terms, SOCKS 4A functions as an independent protocol that occupies the same version number space while introducing semantic behaviors that directly conflict with the original specification. While SOCKSv4 mandates that the client perform destination address resolution prior to the request—thereby transmitting a definitive four-octet IPv4 address—SOCKS 4A introduces a mechanism where the server assumes responsibility for DNS resolution. This shift is triggered by a specific heuristic where the client provides an invalid IP address in the format 0.0.0.x.

This intersection creates a significant protocol conflict. According to the original SOCKSv4 design, a server receiving a destination IP in the 0.0.0.x range should treat it as a literal, albeit unreachable, network address and immediately reject the request with a failure code. However, a SOCKS 4A-compliant server intercepts this specific bit pattern to signal the presence of a trailing variable-length domain name field located after the initial null-terminated user identifier. This divergence means that a standard SOCKSv4 server cannot safely ignore the additional data appended by a 4A client; doing so would result in the trailing domain name being misinterpreted as the start of the application data stream or causing a synchronization error in the TCP buffer. Consequently, SOCKS 4A should be treated as a distinct experimental branch that successfully gained market dominance due to the practical necessity of server-side resolution in environments where clients lack direct DNS access, such as those described in the context of private addressing in [RFC1918].

Appendix B. Operational Considerations

The following sections provide a analysis of the operational realities of SOCKS version 4, accounting for its historical evolution and the practical interpretations that have shaped its deployment in modern network environments.

B.1. Operational Security and Access Control in BIND

In practical deployments, the BIND operation deviates significantly from its theoretical description as a simple port-binding utility. While the protocol fields DSTIP and DSTPORT are nominally intended to identify the target application server, most production-grade SOCKS implementations utilize these fields as a primitive form of an Access Control List (ACL). This behavior is driven by the security requirements outlined in RFC 3552, as allowing an unrestricted

inbound socket on a firewall host would present an unacceptable internal network risk. Most servers enforce a strict source-address restriction, ensuring that the incoming connection to the temporary listening port originates specifically from the IP address provided in the client's initial BIND request.

Furthermore, the operational stability of the BIND command is often compromised by the presence of Network Address Translation (NAT) devices between the SOCKS server and the application server. If the application server's perceived IP address changes due to a NAT gateway, the SOCKS server's security check will fail, returning a rejection code despite a legitimate connection attempt. While some implementations attempt to extend this verification to the source port (DSTPORT), this is widely considered an unreliable practice. As per the transport layer behaviors defined in [RFC9293], source ports for outbound connections are typically ephemeral and allocated dynamically by the operating system's stack, making them unpredictable for the purpose of pre-configured access control.

B.2. Implementation of State Management and Timeouts

The lack of explicit state-tracking mechanisms in the SOCKSv4 control plane necessitates the implementation of aggressive timeout policies to prevent resource exhaustion. Standard operational practice involves two distinct timer categories: the connection establishment timer and the idle listener timer. For CONNECT operations, servers typically implement a 120-second limit for the three-way handshake with the destination host. Failure to receive an ACK within this window results in a code 91 reply and immediate teardown of the client-to-proxy segment. This prevents the "hanging" of file descriptors which could be exploited in a low-bandwidth Denial of Service attack as categorized in [RFC4732].

The BIND operation introduces a more complex state requirement. After the SOCKS server sends the initial success reply containing its local listening port, it must maintain an open socket waiting for the application server's secondary connection. Modern implementations strictly limit this "waiting" state to a narrow window, often significantly shorter than the default TCP timeout. If the expected inbound SYN packet does not arrive within this period, the server must abort the BIND process to free the ephemeral port for other users. This rigorous management of the listener state is a critical operational safeguard, ensuring that a single misbehaving or malicious client cannot monopolize the proxy's available port range or impact the overall availability of the gateway service.

Appendix C. Security Analysis

The SOCKS Version 4 (SOCKSv4) protocol was designed as a pragmatic mechanism for TCP proxying across firewalls in an era when the Internet threat model was significantly less hostile than at present. By contemporary standards, as established in [RFC3552] and [RFC3365], SOCKSv4 is considered fundamentally insecure. It fails to meet the "strong security" requirements mandated for Internet protocols because it lacks native mechanisms for mutual authentication, data confidentiality, and integrity protection.

C.1. Weaknesses in Identification and Authentication

The primary mechanism for client identification in SOCKSv4 is the USERID field, typically leveraged in conjunction with the Identification Protocol (IDENT) as defined in [RFC1413]. As noted in the security considerations of [RFC1413], the information returned by an IDENT server is only as trustworthy as the client host and the network path. In a modern decentralized network, a malicious actor can easily spoof IDENT responses or disable the service entirely, rendering the USERID field unsuitable for any meaningful authorization decisions. Furthermore, SOCKSv4 provides no facility for server-to-client authentication, leaving the client vulnerable to "rogue proxy" attacks where an adversary intercepts the connection and masquerades as the intended SOCKS server. This lack of cryptographic authentication deviates from the best practices for session-layer protocols outlined in [RFC1928].

C.2. Absence of Confidentiality and Integrity

SOCKSv4 operates strictly as a plaintext relay. It does not incorporate any cryptographic transforms to protect the application data stream. Consequently, all traffic traversing a SOCKSv4 proxy is susceptible to passive eavesdropping and active injection or modification by any entity with access to the network path. Under the criteria defined in BCP 61 [RFC3365], protocols that fail to implement strong encryption are insufficient for use over the public Internet. While SOCKSv4 is confined to proxying TCP connections as defined in [RFC9293], its inability to handle UDP traffic (defined in [RFC768]) or provide per-packet integrity checks means that even the protocol's control plane—such as the BIND and CONNECT requests—can be manipulated by an on-path attacker.

C.3. Vulnerabilities in the BIND Operation

The BIND command, intended to support protocols requiring secondary inbound connections, presents a significant attack surface. The protocol's reliance on a rudimentary source IP address check to validate the incoming "callback" connection is inherently flawed. As documented in [RFC1948] and [RFC4953], IP address-based authentication is easily subverted through IP spoofing. Additionally, the prevalence of Network Address Translation (NAT) and middleboxes in modern architectures frequently masks the true source IP of the remote host, making the SOCKSv4 BIND verification either operationally brittle or entirely ineffective. An attacker can exploit this weakness to hijack the inbound socket, potentially gaining unauthorized access to the client's internal network environment.

C.4. Susceptibility to Resource Exhaustion

SOCKSv4 lacks robust flow control and state management for its control plane, making it a viable vector for Denial of Service (DoS) attacks. Every request, particularly the BIND operation which requires the server to listen for an indeterminate period, consumes finite system resources including memory, file descriptors, and kernel state. While the protocol suggests a two-minute timeout for connection establishment, this fixed value is not an adequate defense against coordinated resource exhaustion attacks. Without the modern rate-limiting and state-tracking mechanisms discussed in [RFC4732], a SOCKSv4 server can be easily overwhelmed by a relatively small number of malicious clients.

C.5. Deployment Limitations and Mitigation

Given the deficiencies detailed above, SOCKSv4 is classified as a "Historic" protocol and its use is strongly discouraged. In scenarios where legacy requirements necessitate its deployment, it MUST be encapsulated within a secure transport layer, such as Transport Layer Security (TLS) defined in [RFC8446] or an IPsec tunnel as defined in [RFC4301], to provide the requisite security properties. Operators are urged to migrate to SOCKS Version 5 [RFC1928], which supports extensible authentication (GSS-API, etc.) and UDP proxying, or to modern proxying solutions that satisfy the security requirements of the IETF "Danvers Doctrine" as memorialized in [RFC3365].

Appendix D. Existing Values

The existing values used within the protocol are summarized below:

D.1. SOCKS Protocol Version Number

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

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

D.2. SOCKS Command Code

The SOCKS command code CD in requests defines two values:

* 1 (0x01): CONNECT

* 2 (0x02): BIND

D.3. SOCKS Reply Code

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

D.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 Author

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