Network Working Group
Internet-Draft
Updates: 1928 (if approved)
Intended status: Standards Track
Expires: 7 October 2026

D. J. Vance
Independent

5 April 2026

Deprecating the FRAG Field in SOCKS5
draft-vance-socks5-frag-deprecation-00

Abstract

This document updates RFC 1928 by formally deprecating the FRAG (Fragment) field in the SOCKS5 UDP ASSOCIATE request header. It mandates that the FRAG field MUST be set to X'00' by clients and that proxies MUST drop any SOCKS5 UDP packets containing a non-zero FRAG value. This change aligns the SOCKS5 protocol with modern Internet engineering practices regarding UDP fragmentation (BCP 227), simplifies proxy implementations, and eliminates a vector for resource exhaustion attacks.

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/socksbis/frag.

Status of This Memo

Table of Contents

1. Introduction

The SOCKS Protocol Version 5 [RFC1928] defines a framework for client-server communication through a proxy. Section 7 of [RFC1928] describes a mechanism for UDP application-level fragmentation, utilizing a "FRAG" field in the SOCKS UDP request header to split large datagrams.

Since 1996, the operational reality of the Internet has evolved. IP fragmentation is now recognized as fragile and detrimental to performance and security [RFC8900]. Modern UDP applications are expected to handle Maximum Transmission Unit (MTU) discovery independently [RFC8085] [RFC8899].

This document updates [RFC1928] by deprecating the SOCKS5 UDP fragmentation mechanism to improve security and interoperability.

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.

3. Background: The SOCKS5 FRAG Field

Section 7 of [RFC1928] defines a mechanism for application-level fragmentation within the SOCKS UDP relay. The one-octet FRAG field is used to indicate the position of a datagram within a fragment sequence. A value of X’00’ signifies that the datagram is standalone, while values from 1 to 127 represent the fragment’s position. The high-order bit of the field serves as an indicator for the end of a fragment sequence.

The original specification stipulates that any receiver implementing this feature maintains a reassembly queue and a reassembly timer for these fragments. This timer is required to be no less than five seconds. The reassembly state is reset and any associated fragments are abandoned if the timer expires or if an incoming datagram carries a FRAG value lower than the highest value previously processed for that sequence. Despite the inclusion of this mechanism, [RFC1928] explicitly recommends that applications avoid fragmentation whenever possible.

Furthermore, the implementation of this fragmentation mechanism is designated as optional. Proxies that do not support the feature are expected to drop any datagram containing a FRAG field value other

than X’00’. The protocol also includes specific requirements for SOCKS-aware UDP programming interfaces, which are expected to report a smaller available buffer space to applications to account for the overhead of the SOCKS header. The exact reduction in reported buffer space depends on the address type (ATYP) used, ranging from 10 to over 262 octets plus method-dependent overhead.

4. Motivation for Deprecation

4.1. Statefulness and Resource Exhaustion

UDP is architecturally defined as a stateless, best-effort transport layer. By requiring a SOCKS5 proxy to implement Section 7 of [RFC1928], the protocol introduces an application-layer reassembly subsystem that fundamentally contradicts the stateless nature of UDP. This mandatory state management for unauthenticated or semi-authenticated flows presents several distinct vectors for service degradation and system failure.

4.1.1. Incomplete Sequence Memory Exhaustion

An attacker may initiate a high volume of UDP ASSOCIATE flows, sending only a subset of fragments for each sequence (e.g., sending fragment X’01’ of X’05’ and never transmitting the subsequent octets). To comply with [RFC1928], the proxy must allocate a reassembly buffer for each unique fragment sequence, indexed by the source address, port, and destination metadata.

Since the proxy cannot forward the datagram until the sequence is contiguous, it must retain these partial payloads in memory. By rotating source ports or utilizing spoofed source addresses (where ingress filtering is absent), an attacker can rapidly deplete the proxy’s heap memory or specialized reassembly buffers. This results in a Denial-of-Service (DoS) condition affecting not only UDP traffic but potentially the entire proxy process or host system.

4.1.2. Timer Management and CPU Overload

The requirement to maintain reassembly timers introduces additional computational overhead. Each fragmented flow necessitates a tracking mechanism to expire incomplete sequences and reclaim memory. In a high-concurrency environment, an attacker can oscillate fragment transmission rates to force the proxy into frequent timer-eviction cycles. The overhead of managing thousands of concurrent timers— specifically the sorting and searching of timer heaps or linked lists— can lead to significant CPU exhaustion, increasing latency for legitimate proxied connections.

4.1.3. Fragment Overlap and Algorithmic Complexity Attacks

Similar to historical vulnerabilities in the IP stack, the SOCKS5 fragmentation field allows for the transmission of overlapping or out-of-order application-layer fragments. If a proxy implementation does not strictly validate fragment boundaries, an attacker may send fragments with overlapping sequence numbers to trigger expensive memory-copy operations or to exploit edge cases in the reassembly logic. A malicious actor could craft sequences that require the proxy to perform repeated buffer reallocations or complex data-shifting operations, leading to an algorithmic complexity attack that degrades performance disproportionately to the attacker’s bandwidth expenditure.

4.1.4. Pre-authentication Vulnerability

A critical weakness in the SOCKS5 fragmentation mechanism is its availability prior to, or during, the establishment of a fully authenticated session. Because UDP ASSOCIATE may be utilized in environments with minimal authentication, or because the reassembly logic often sits at the ingress of the UDP relay, the proxy is forced to expend resources on behalf of unverified actors. This asymmetry allows a low-resource attacker to impose high-resource costs on the infrastructure, making the FRAG field a primary vector for unauthenticated resource exhaustion.

4.2. Alignment with BCP 227

As documented in [RFC8900], middleboxes frequently drop fragmented packets. Reassembling fragments at the proxy and forwarding a large egress datagram often triggers IP fragmentation, leading to high packet loss. End-to-end MTU management [RFC8899] is the current architectural best practice.

4.3. Implementation Status

Modern, production-grade SOCKS5 implementations have largely abandoned support for Section 7 of [RFC1928]. Formalizing this deprecation prevents interoperability issues with legacy or non-compliant implementations.

5. Normative Changes to RFC 1928

This document updates Section 7 of [RFC1928].

5.1. Client Requirements

SOCKS5 clients MUST set the FRAG field to X’00’ in all transmitted UDP packets.

Clients MUST NOT attempt to fragment UDP datagrams at the SOCKS protocol layer. If a datagram exceeds the path MTU, the client SHOULD employ Path MTU Discovery mechanisms (e.g., PLPMTUD [RFC8899]) to adjust the payload size before encapsulation.

5.2. Proxy Server Requirements

A SOCKS5 proxy server MUST NOT maintain reassembly queues or timers for UDP fragments.

Upon receiving a UDP packet with a FRAG field not equal to X’00’, the proxy server MUST silently discard the packet. The proxy MUST NOT generate any signaling or error responses (e.g., ICMP unreachable messages or SOCKS-level control notifications) in response to such packets. This prevents the proxy from acting as a reflection or amplification vector in Denial-of-Service attacks. For observability, implementations SHOULD provide internal counters to track the number of packets dropped due to non-zero FRAG values.

5.3. Revised UDP Request Header Format

To reflect the deprecation of the fragmentation mechanism, the SOCKS UDP request header is updated as follows. The field previously known as FRAG is now designated as RSV (Reserved) and MUST be ignored for its original purpose.

Field	Size (Octets)	Description
RSV	2	Reserved (X’0000’)
FRAG	1	DEPRECATED. Current implementations MUST set to X’00’.
ATYP	1	Address type of following address
DST.ADDR	Variable	Desired destination address
DST.PORT	2	Desired destination port
DATA	Variable	User data

Table 1: Revised UDP Request Header Format

6. Compatibility Considerations

The deprecation of the FRAG field is designed to have minimal impact on the existing installed base of SOCKS5 implementations, as it formalizes the de facto behavior of the majority of modern deployments.

6.1. Legacy Proxy Interoperability

According to Section 7 of [RFC1928], the implementation of fragmentation reassembly was designated as OPTIONAL. Consequently, robust SOCKS5 clients have historically been required to handle scenarios where a proxy might drop fragments. A compliant client implemented according to this update will set the FRAG field to X’00’, which is perfectly backward compatible with any legacy proxy server, whether or not that legacy proxy supported fragmentation.

6.2. Impact on Legacy Clients

Legacy clients that rely on SOCKS-layer fragmentation (sending non-zero FRAG values) will experience a loss of UDP connectivity when communicating through a proxy server compliant with this document. However, empirical observation of Internet traffic suggests that SOCKS-layer fragmentation is extremely rare in modern production environments. Most high-performance proxies have already ceased supporting fragmentation due to the resource exhaustion risks outlined in Section 4.

Furthermore, since [RFC1928] did not provide a mechanism for a client to negotiate or discover if a proxy supported fragmentation, any client relying on this feature was already operating in a fragile state. By mandating the drop of fragmented packets, this document provides a predictable failure mode that encourages application developers to implement robust end-to-end MTU management at the packetization layer [RFC8899], rather than relying on fragile middlebox reassembly.

6.3. The Robustness Principle

While the "Robustness Principle" suggests being liberal in what you accept, modern security considerations [RFC9413] increasingly favor strict protocol enforcement to eliminate ambiguity and reduce attack surface. The deprecation of the FRAG field is a deliberate trade-off, prioritizing system stability, security, and protocol simplicity over the support of a legacy feature that is widely considered detrimental to modern network performance.

7. Operational Considerations

Operational visibility is essential for distinguishing between legitimate legacy client traffic and potential denial-of-service attacks. While this document mandates the silent discard of fragmented UDP packets to prevent amplification, it is critical that operators can monitor these events.

Implementations SHOULD maintain telemetry counters to track the frequency and volume of packets discarded due to non-zero FRAG values. These counters provide low-overhead observability and should be integrated into existing monitoring frameworks (e.g., SNMP, YANG, or vendor-specific telemetry).

For more granular troubleshooting, implementations MAY provide logging mechanisms to identify the source addresses of misconfigured legacy clients. However, to mitigate the risk of log-exhaustion attacks, such logging MUST be strictly rate-limited. Excessive logging can consume significant CPU and storage resources, potentially leading to a secondary denial-of-service condition. Operators are advised to use these logs primarily for short-term diagnostic purposes rather than continuous monitoring in high-traffic environments.

8. Security Considerations

This document enhances the security of SOCKS5 deployments. By removing the reassembly logic, the attack surface for memory corruption vulnerabilities (e.g., buffer overflows) is significantly reduced.

Furthermore, prohibiting error responses for dropped fragments ensures that the SOCKS proxy cannot be used as an amplification vector in a Distributed Denial of Service (DDoS) attack.

9. IANA Considerations

This document has no IANA actions.

10. References

10.1. Normative 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/info/rfc1928>.

[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/info/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/info/rfc8174>.

10.2. Informative References

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[RFC8899] Fairhurst, G., Jones, T., Txen, M., Rngeler, I., and T. Vlker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

[RFC8900] Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O., and F. Gont, "IP Fragmentation Considered Fragile", BCP 230, RFC 8900, DOI 10.17487/RFC8900, September 2020, <https://www.rfc-editor.org/info/rfc8900>.

[RFC9413] Thomson, M. and D. Schinazi, "Maintaining Robust Protocols", RFC 9413, DOI 10.17487/RFC9413, June 2023, <https://www.rfc-editor.org/info/rfc9413>.

Appendix A. Acknowledgments

The author wishes to thank Michael Richardson for their valuable feedback and suggestions during the discussion of this document on the IETF mailing lists.

Author’s Address

Daniel James Vance
Independent
Email: djvanc@outlook.com