Stream:	Internet Engineering Task Force (IETF)	
RFC:	9289	
Updates:	5531	
Category:	Standards Track	
Published:	September 2022	
ISSN:	2070-1721	
Authors:	T. Myklebust	C. Lever, Ed.
	Hammerspace	*Oracle*

RFC 9289
Towards Remote Procedure Call Encryption by Default

Abstract

This document describes a mechanism that, through the use of opportunistic Transport Layer Security (TLS), enables encryption of Remote Procedure Call (RPC) transactions while they are in transit. The proposed mechanism interoperates with Open Network Computing (ONC) RPC implementations that do not support it. This document updates RFC 5531.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9289.

Copyright Notice

Table of Contents

1. Introduction

In 2014 the IETF published a document entitled "Pervasive Monitoring Is an Attack" [RFC7258], which recognized that unauthorized observation of network traffic had become widespread and was a subversive threat to all who make use of the Internet at large. It strongly recommended that newly defined Internet protocols should make a genuine effort to mitigate monitoring attacks. Typically, this mitigation includes encrypting data in transit.

The Remote Procedure Call version 2 protocol has been a Proposed Standard for three decades (see [RFC5531] and its antecedents). Over twenty years ago, Eisler et al. first introduced RPCSEC_GSS as an in-transit encryption mechanism for RPC [RFC2203]. However, experience has shown that RPCSEC_GSS with in-transit encryption can be challenging to use in practice due to the following:

- Parts of each RPC header remain in cleartext, constituting a loss of metadata confidentiality.
- Offloading the Generic Security Service (GSS) privacy service is not practical in large multi-user deployments since each message is encrypted using a key based on the issuing RPC user.

However strong GSS-provided confidentiality is, it cannot provide any security if the challenges of using it result in choosing not to deploy it at all.

Moreover, the use of AUTH_SYS remains common despite the adverse effects that acceptance of User Identifiers (UIDs) and Group Identifiers (GIDs) from unauthenticated clients brings with it. Continued use is in part because:

- Per-client deployment and administrative costs for the only well-defined alternative to AUTH_SYS are expensive at scale. For instance, administrators must provide keying material for each RPC client, including transient clients.

- GSS host identity management and user identity management typically must be enforced in the same security realm. However, cloud providers, for instance, might prefer to remain authoritative for host identity but allow tenants to manage user identities within their private networks.

In view of the challenges with the currently available mechanisms for authenticating and protecting the confidentiality of RPC transactions, this document specifies a transport-layer security mechanism that complements the existing ones. The TLS [RFC8446] and Datagram Transport Layer Security (DTLS) [RFC9147] protocols are well-established Internet building blocks that protect many standard Internet protocols such as the Hypertext Transfer Protocol (HTTP) [RFC9110].

Encrypting at the RPC transport layer accords several significant benefits:

Encryption by Default: Transport encryption can be enabled without additional administrative tasks such as identifying client systems to a trust authority and providing each with keying material.

Encryption Offload: Hardware support for the GSS privacy service has not appeared in the marketplace. However, the use of a well-established transport encryption mechanism that is employed by other ubiquitous network protocols makes it more likely that encryption offload for RPC is practicable.

Securing AUTH_SYS: Most critically, transport encryption can significantly reduce several security issues inherent in the current widespread use of AUTH_SYS (i.e., acceptance of UIDs and GIDs generated by an unauthenticated client).

Decoupled User and Host Identities: TLS can be used to authenticate peer hosts while other security mechanisms can handle user authentication.

Compatibility: The imposition of encryption at the transport layer protects any upper-layer protocol that employs RPC, without alteration of the upper-layer protocol.

Further, Section 6 of the current document defines policies in line with [RFC7435] that enable RPC-with-TLS to be deployed opportunistically in environments that contain RPC implementations that do not support TLS. However, specifications for RPC-based upper-layer protocols should choose to require even stricter policies that guarantee encryption and host authentication are used for all RPC transactions to mitigate against pervasive monitoring attacks [RFC7258]. Enforcing the use of RPC-with-TLS is of particular importance for existing upper-layer protocols whose security infrastructure is weak.

The protocol specification in the current document assumes that support for ONC RPC [RFC5531], TLS [RFC8446], PKIX [RFC5280], DNSSEC/DNS-Based Authentication of Named Entities (DANE) [RFC6698], and optionally RPCSEC_GSS [RFC2203] is available within the platform where RPC-with-TLS support is to be added.

2. Requirements Language

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. Terminology

This document adopts the terminology introduced in Section 3 of [RFC6973] and assumes a working knowledge of the RPC version 2 protocol [RFC5531] and the TLS version 1.3 protocol [RFC8446].

Note also that the NFS community long ago adopted the use of the term "privacy" from documents such as [RFC2203]. In the current document, the authors use the term "privacy" only when referring specifically to the historic GSS privacy service defined in [RFC2203]. Otherwise, the authors use the term "confidentiality", following the practices of contemporary security communities.

We adhere to the convention that a "client" is a network host that actively initiates an association, and a "server" is a network host that passively accepts an association request.

RPC documentation historically refers to the authentication of a connecting host as "machine authentication" or "host authentication". TLS documentation refers to the same as "peer authentication". In the current document, there is little distinction between these terms.

The term "user authentication" in the current document refers specifically to the RPC caller's credential, provided in the "cred" and "verf" fields in each RPC Call.

4. RPC-with-TLS in Operation

4.1. Discovering Server-Side TLS Support

The mechanism described in the current document interoperates fully with RPC implementations that do not support RPC-with-TLS. When an RPC-with-TLS-enabled peer encounters a peer that does not support RPC-with-TLS, policy settings on the RPC-with-TLS-enabled peer determine whether RPC operation continues without the use of TLS or is discontinued altogether.

To achieve this interoperability, we introduce a new RPC authentication flavor called AUTH_TLS. The AUTH_TLS authentication flavor signals that the client wants to initiate TLS negotiation if the server supports it. Except for the modifications described in this section, the RPC protocol is unaware of security encapsulation at the transport layer. The value of AUTH_TLS is defined in Section 7.1.

An RPC client begins its communication with an RPC server by selecting a transport and destination port. The choice of transport and port is typically based on the RPC program that is to be used. The RPC client might query the RPC server's RPCBIND service to make this selection (The RPCBIND service is described in [RFC1833]). The mechanism described in the current document does not support RPC transports other than TCP and UDP. In all cases, an RPC server **MUST** listen on the same ports for (D)TLS-protected RPC programs as the ports used when (D)TLS is not available.

To protect RPC traffic to a TCP port, the RPC client opens a TCP connection to that port and sends a NULL RPC procedure with an auth_flavor of AUTH_TLS on that connection. To protect RPC traffic to a UDP port, the RPC client sends a UDP datagram to that port containing a NULL RPC procedure with an auth_flavor of AUTH_TLS. The client constructs this RPC procedure as follows:

- The length of the opaque data constituting the credential sent in the RPC Call message **MUST** be zero.
- The verifier accompanying the credential **MUST** be an AUTH_NONE verifier of length zero.
- The flavor value of the verifier in the RPC Reply message received from the server **MUST** be AUTH_NONE.
- The length of the verifier's body field is eight.
- The bytes of the verifier's body field encode the ASCII characters "STARTTLS" as a fixed-length opaque.

The RPC server signals its corresponding support for RPC-with-TLS by replying with a reply_stat of MSG_ACCEPTED and an AUTH_NONE verifier containing the "STARTTLS" token. The client **SHOULD** proceed with TLS session establishment, even if the Reply's accept_stat is not SUCCESS. If the AUTH_TLS probe was done via TCP, the RPC client **MUST** send the "ClientHello" message on the same connection. If the AUTH_TLS probe was done via UDP, the RPC client **MUST** send the "ClientHello" message to the same UDP destination port.

Conversely, if the Reply's reply_stat is not MSG_ACCEPTED, if its verifier flavor is not AUTH_NONE, or if its verifier does not contain the "STARTTLS" token, the RPC client **MUST NOT** send a "ClientHello" message. RPC operation may continue, depending on local policy, but without confidentiality, integrity, or peer authentication protection from (D)TLS.

If, after a successful RPC AUTH_TLS probe, the subsequent (D)TLS handshake should fail for any reason, the RPC client reports this failure to the upper-layer application the same way it reports an AUTH_ERROR rejection from the RPC server.

If an RPC client uses the AUTH_TLS authentication flavor on any procedure other than the NULL procedure, or an RPC client sends an RPC AUTH_TLS probe within an existing (D)TLS session, the RPC server **MUST** reject that RPC Call by returning a reply_stat of MSG_DENIED with a reject_stat of AUTH_ERROR and an auth_stat of AUTH_BADCRED.

Once the TLS session handshake is complete, the RPC client and server have established a secure channel for exchanging RPC transactions. A successful AUTH_TLS probe on one particular port/transport tuple does not imply that RPC-with-TLS is available on that same server using a different port/transport tuple, nor does it imply that RPC-with-TLS will be available in the future using the successfully probed port.

4.2. Authentication

There is some overlap between the authentication capabilities of RPC and TLS. The goal of interoperability with implementations that do not support TLS requires limiting the combinations that are allowed and precisely specifying the role that each layer plays.

Each RPC server that supports RPC-with-TLS **MUST** possess a unique global identity (e.g., a certificate that is signed by a well-known trust anchor). Such an RPC server **MUST** request a TLS peer identity from each client upon first contact. There are two different modes of client deployment:

Server-Only Host Authentication
: In this type of deployment, the client can authenticate the server host using the presented server peer TLS identity, but the server cannot authenticate the client. In this situation, RPC-with-TLS clients are anonymous. They present no globally unique identifier to the server peer.

Mutual Host Authentication
: In this type of deployment, the client possesses an identity that is backed by a trusted entity (e.g., a pre-shared key or a certificate validated with a certification path). As part of the TLS handshake, both peers authenticate using the presented TLS identities. If authentication of either peer fails, or if authorization based on those identities blocks access to the server, the peers **MUST** reject the association. Further explanation appears in Section 5.2.

In either of these modes, RPC user authentication is not affected by the use of transport layer security. When a client presents a TLS peer identity to an RPC server, the protocol extension described in the current document provides no way for the server to know whether that identity represents one RPC user on that client or is shared amongst many RPC users. Therefore, a server implementation cannot utilize the remote TLS peer identity to authenticate RPC users.

4.2.1. Using TLS with RPCSEC_GSS

To use GSS, an RPC server has to possess a GSS service principal. On a TLS session, GSS mutual (peer) authentication occurs as usual, but only after a TLS session has been established for communication. Authentication of RPCSEC_GSS users is unchanged by the use of TLS.

RPCSEC_GSS can also perform per-request integrity or confidentiality protection. When operating over a TLS session, these GSS services become largely redundant. An RPC implementation capable of concurrently using TLS and RPCSEC_GSS **MUST** use Generic Security Service Application Program Interface (GSS-API) channel binding, as defined in [RFC5056], to

determine when an underlying transport provides a sufficient degree of confidentiality. RPC-with-TLS implementations **MUST** provide the "tls-exporter" channel binding type, as defined in [RFC9266].

5. TLS Requirements

When peers negotiate a TLS session that is to transport RPC, the following restrictions apply:

- Implementations **MUST NOT** negotiate TLS versions prior to 1.3 (for TLS [RFC8446] or DTLS [RFC9147], respectively). Support for mandatory-to-implement cipher suites for the negotiated TLS version is **REQUIRED**.
- Implementations **MUST** conform to the recommendations for TLS usage specified in BCP 195 [RFC7525]. Although RFC 7525 permits the use of TLS 1.2, the requirement to use TLS 1.3 or later for RPC-with-TLS takes precedence. Further, because TLS 1.3 ciphers are qualitatively different than cipher suites in previous versions of TLS, and RFC 7525 predates TLS 1.3, the cipher suite recommendations in RFC 7525 do not apply to RPC-with-(D)TLS. A strict TLS mode for RPC-with-TLS that protects against STRIPTLS attacks is discussed in detail in Section 6.1.1.
- Implementations **MUST** support certificate-based mutual authentication. Support for Pre-Shared Key (PSK) mutual authentication is **OPTIONAL**; see Section 5.2.2 for further details.
- Negotiation of a cipher suite providing confidentiality as well as integrity protection is **REQUIRED**.

Client implementations **MUST** include the "application_layer_protocol_negotiation(16)" extension [RFC7301] in their "ClientHello" message and **MUST** include the protocol identifier defined in Section 7.2 in that message's ProtocolNameList value.

Similarly, in response to the "ClientHello" message, server implementations **MUST** include the "application_layer_protocol_negotiation(16)" extension [RFC7301] in their "ServerHello" message and **MUST** include only the protocol identifier defined in Section 7.2 in that message's ProtocolNameList value.

If the server responds incorrectly (for instance, if the "ServerHello" message does not conform to the above requirements), the client **MUST NOT** establish a TLS session for use with RPC on this connection. See [RFC7301] for further details about how to form these messages properly.

5.1. Base Transport Considerations

There is frequently a strong association between an RPC program and a particular destination port number. The use of TLS or DTLS does not change that association. Thus, it is frequently, though not always, the case that a single TLS session carries traffic for only one RPC program.

5.1.1. Protected Operation on TCP

The use of the TLS protocol [RFC8446] protects RPC on TCP connections. Typically, once an RPC client completes the TCP handshake, it uses the mechanism described in Section 4.1 to discover RPC-with-TLS support for that RPC program on that connection. Until an AUTH_TLS probe is

done on a connection, the RPC server treats all traffic as RPC messages. If spurious traffic appears on a TCP connection between the initial cleartext AUTH_TLS probe and the TLS session handshake, receivers **MUST** discard that data without response and then **SHOULD** drop the connection.

The protocol convention specified in the current document assumes there can be no more than one concurrent TLS session per TCP connection. This is true of current generations of TLS, but might be different in a future version of TLS.

Once a TLS session is established on a TCP connection, no further cleartext communication can occur on that connection until the session is terminated. The use of TLS does not alter RPC record framing used on TCP transports.

Furthermore, if an RPC server responds with PROG_UNAVAIL to an RPC Call within an established TLS session, that does not imply that RPC server will subsequently reject the same RPC program on a different TCP connection.

Reverse-direction operation occurs only on connected transports such as TCP (see Section 2 of [RFC8167]). To protect reverse-direction RPC operations, the RPC server does not establish a separate TLS session on the TCP connection but instead uses the existing TLS session on that connection to protect these operations.

When operation is complete, an RPC peer terminates a TLS session by sending a TLS closure alert. It may then close the TCP connection.

5.1.2. Protected Operation on UDP

The use of the DTLS protocol [RFC9147] protects RPC carried in UDP datagrams. As soon as a client initializes a UDP socket for use with an RPC service, it uses the mechanism described in Section 4.1 to discover RPC-with-DTLS support for that RPC program on that port. If spurious traffic appears on a 5-tuple between the initial cleartext AUTH_TLS probe and the DTLS association handshake, receivers **MUST** discard that traffic without response.

Using DTLS does not introduce reliable or in-order semantics to RPC on UDP. The use of DTLS record replay protection is **REQUIRED** when transporting RPC traffic.

Each RPC message **MUST** fit in a single DTLS record. DTLS encapsulation has overhead, which reduces the Packetization Layer Path MTU (PLPMTU) and thus the maximum RPC payload size. A possible PLPMTU discovery mechanism is offered in [RFC8899].

The current document does not specify a mechanism that enables a server to distinguish between DTLS traffic and unprotected RPC traffic directed to the same port. To make this distinction, each peer matches ingress datagrams that appear to be DTLS traffic to existing DTLS session state. A peer treats any datagram that fails the matching process as an RPC message.

Multihomed RPC clients and servers may send protected RPC messages via network interfaces that were not involved in the handshake that established the DTLS session. Therefore, when protecting RPC traffic, each DTLS handshake **MUST** include the "connection_id(54)" extension described in Section 9 of [RFC9147], and RPC-with-DTLS peer endpoints **MUST** provide a

ConnectionID with a nonzero length. Endpoints implementing RPC programs that expect a significant number of concurrent clients **SHOULD** employ ConnectionIDs of at least 4 bytes in length.

Sending a TLS closure alert terminates a DTLS session. Because neither DTLS nor UDP provide in-order delivery, after session closure there can be ambiguity as to whether a datagram should be interpreted as DTLS protected or not. Therefore, receivers **MUST** discard datagrams exchanged using the same 5-tuple that just terminated the DTLS session for a sufficient length of time to ensure that retransmissions have ceased and packets already in the network have been delivered. In the absence of more specific data, a period of 60 seconds is expected to suffice.

5.1.3. Protected Operation on Other Transports

Transports that provide intrinsic TLS-level security (e.g., QUIC) need to be addressed separately from the current document. In such cases, the use of TLS is not opportunistic as it can be for TCP or UDP.

RPC-over-RDMA can make use of transport layer security below the RDMA transport layer [RFC8166]. The exact mechanism is not within the scope of the current document. Because there might not be other provisions to exchange client and server certificates, authentication material exchange needs to be provided by facilities within a future version of the RPC-over-RDMA transport protocol.

5.2. TLS Peer Authentication

TLS can perform peer authentication using any of the following mechanisms.

5.2.1. X.509 Certificates Using PKIX Trust

X.509 certificates are specified in [X.509]. [RFC5280] provides a profile of Internet PKI X.509 public key infrastructure. RPC-with-TLS implementations are **REQUIRED** to support the PKIX mechanism described in [RFC5280].

The rules and guidelines defined in [RFC6125] apply to RPC-with-TLS certificates with the following considerations:

- The DNS-ID identifier type is a subjectAltName extension that contains a dNSName, as defined in Section 4.2.1.6 of [RFC5280]. Support for the DNS-ID identifier type is **REQUIRED** in RPC-with-TLS client and server implementations. Certification authorities that issue such certificates **MUST** support the DNS-ID identifier type.
- To specify the identity of an RPC peer as a domain name, the certificate **MUST** contain a subjectAltName extension that contains a dNSName. DNS domain names in RPC-with-TLS certificates **MUST NOT** contain the wildcard character '*' within the identifier.
- To specify the identity of an RPC peer as a network identifier (netid) or a universal network address (uaddr), the certificate **MUST** contain a subjectAltName extension that contains an iPAddress.

When validating a server certificate, an RPC-with-TLS client implementation takes the following into account:

- Certificate validation **MUST** include the verification rules as per Section 6 of [RFC5280] and Section 6 of [RFC6125].
- Server certificate validation **MUST** include a check on whether the locally configured expected DNS-ID or iPAddress subjectAltName of the server that is contacted matches its presented certificate.
- For RPC services accessed by their netids and uaddrs, the iPAddress subjectAltName **MUST** be present in the certificate and **MUST** exactly match the address represented by the universal network address.

An RPC client's domain name and IP address are often assigned dynamically; thus, RPC servers cannot rely on those to verify client certificates. Therefore, when an RPC-with-TLS client presents a certificate to an RPC-with-TLS server, the server takes the following into account:

- The server **MUST** use a procedure conformant to Section 6 of [RFC5280] to validate the client certificate's certification path.
- The tuple (serial number of the presented certificate; Issuer) uniquely identifies the RPC client. The meaning and syntax of these fields is defined in Section 4 of [RFC5280].

RPC-with-TLS implementations **MAY** allow the configuration of a set of additional properties of the certificate to check for a peer's authorization to communicate (e.g., a set of allowed values in subjectAltName:URI, a set of allowed X.509v3 Certificate Policies, or a set of extended key usages).

When the configured set of trust anchors changes (e.g., removal of a Certification Authority (CA) from the list of trusted CAs; issuance of a new Certificate Revocation List (CRL) for a given CA), implementations **SHOULD** reevaluate the certificate originally presented in the context of the new configuration and terminate the TLS session if the certificate is no longer trustworthy.

5.2.1.1. Extended Key Usage Values

Section 4.2.1.12 of [RFC5280] specifies the extended key usage X.509 certificate extension. This extension, which may appear in end-entity certificates, indicates one or more purposes for which the certified public key may be used in addition to or in place of the basic purposes indicated in the key usage extension.

The current document defines two new KeyPurposeId values: one that identifies the RPC-with-TLS peer as an RPC client, and one that identifies the RPC-with-TLS peer as an RPC server.

The inclusion of the RPC server value (id-kp-rpcTLSServer) indicates that the certificate has been issued for allowing the holder to process RPC transactions.

The inclusion of the RPC client value (id-kp-rpcTLSClient) indicates that the certificate has been issued for allowing the holder to request RPC transactions.

5.2.2. Pre-shared Keys

This mechanism is **OPTIONAL** to implement. In this mode, the RPC peer can be uniquely identified by keying material that has been shared out of band (see Section 2.2 of [RFC8446]). The PSK Identifier **SHOULD** be exposed at the RPC layer.

6. Security Considerations

One purpose of the mechanism described in the current document is to protect RPC-based applications against threats to the confidentiality of RPC transactions and RPC user identities. A taxonomy of these threats appears in Section 5 of [RFC6973]. Also, Section 6 of [RFC7525] contains a detailed discussion of technologies used in conjunction with TLS. Section 8 of [RFC5280] covers important considerations about handling certificate material securely. Implementers should familiarize themselves with these materials.

Once a TLS session is established, the RPC payload carried on TLS version 1.3 is forward secure. However, implementers need to be aware that replay attacks can occur during session establishment. Remedies for such attacks are discussed in detail in Section 8 of [RFC8446]. Further, the current document does not provide a profile that defines the use of 0-RTT data (see Appendix E.5 of [RFC8446]). Therefore, RPC-with-TLS implementations **MUST NOT** use 0-RTT data.

6.1. The Limitations of Opportunistic Security

Readers can find the definition of Opportunistic Security in [RFC7435]. A discussion of its underlying principles appears in Section 3 of that document.

The purpose of using an explicitly opportunistic approach is to enable interoperation with implementations that do not support RPC-with-TLS. A range of options is allowed by this approach, from "no peer authentication or encryption" to "server-only authentication with encryption" to "mutual authentication with encryption". The actual security level may indeed be selected based on policy and without user intervention.

In environments where interoperability is a priority, the security benefits of TLS are partially or entirely waived. Implementations of the mechanism described in the current document must take care to accurately represent to all RPC consumers the level of security that is actually in effect, and are **REQUIRED** to provide an audit log of RPC-with-TLS security mode selection.

In all other cases, the adoption, implementation, and deployment of RPC-based upper-layer protocols that enforce the use of TLS authentication and encryption (when similar RPCSEC_GSS services are not in use) is strongly encouraged.

6.1.1. STRIPTLS Attacks

The initial AUTH_TLS probe occurs in cleartext. An on-path attacker can alter a cleartext handshake to make it appear as though TLS support is not available on one or both peers. Client implementers can choose from the following to mitigate STRIPTLS attacks:

- A TLSA record [RFC6698] can alert clients that TLS is expected to work, and provide a binding of a hostname to the X.509 identity. If TLS cannot be negotiated or authentication fails, the client disconnects and reports the problem. When an opportunistic security policy is in place, a client **SHOULD** check for the existence of a TLSA record for the target server before initiating an RPC-with-TLS association.
- Client security policy can require that a TLS session is established on every connection. If an attacker spoofs the handshake, the client disconnects and reports the problem. This policy prevents an attacker from causing the association to fall back to cleartext silently. If TLSA records are not available, this approach is strongly encouraged.

6.1.2. Privacy Leakage before Session Establishment

As mentioned earlier, communication between an RPC client and server appears in the clear on the network prior to the establishment of a TLS session. This cleartext information usually includes transport connection handshake exchanges, the RPC NULL procedure probing support for TLS, and the initial parts of TLS session establishment. Appendix C of [RFC8446] discusses precautions that can mitigate exposure during the exchange of connection handshake information and TLS certificate material that might enable attackers to track the RPC client. Note that when PSK authentication is used, the PSK identifier is exposed during the TLS handshake and can be used to track the RPC client.

Any RPC traffic that appears on the network before a TLS session has been established is vulnerable to monitoring or undetected modification. A secure client implementation limits or prevents any RPC exchanges that are not protected.

The exception to this edict is the initial RPC NULL procedure that acts as a STARTTLS message, which cannot be protected. This RPC NULL procedure contains no arguments or results, and the AUTH_TLS authentication flavor it uses does not contain user information, so there is negligible privacy impact from this exception.

6.2. TLS Identity Management on Clients

The goal of RPC-with-TLS is to hide the content of RPC requests while they are in transit. RPC-with-TLS protocol by itself cannot protect against exposure of a user's RPC requests to other users on the same client.

Moreover, client implementations are free to transmit RPC requests for more than one RPC user using the same TLS session. Depending on the details of the client RPC implementation, this means that the client's TLS credentials are potentially visible to every RPC user that shares a TLS session. Privileged users may also be able to access this TLS identity.

As a result, client implementations need to carefully segregate TLS credentials so that local access to it is restricted to only the local users that are authorized to perform operations on the remote RPC server.

6.3. Security Considerations for AUTH_SYS on TLS

Using a TLS-protected transport when the AUTH_SYS authentication flavor is in use addresses several longstanding weaknesses in AUTH_SYS (as detailed in Appendix A). TLS augments AUTH_SYS by providing both integrity protection and confidentiality that AUTH_SYS lacks. TLS protects data payloads, RPC headers, and user identities against monitoring and alteration while in transit.

TLS guards against in-transit insertion and deletion of RPC messages, thus ensuring the integrity of the message stream between RPC client and server. DTLS does not provide full message stream protection, but it does enable receivers to reject nonparticipant messages. In particular, transport-layer encryption plus peer authentication protects receiving eXternal Data Representation (XDR) decoders from deserializing untrusted data, a common coding vulnerability. However, these decoders would still be exposed to untrusted input in the case of the compromise of a trusted peer or Certification Authority.

The use of TLS enables strong authentication of the communicating RPC peers, providing a degree of non-repudiation. When AUTH_SYS is used with TLS, but the RPC client is unauthenticated, the RPC server still acts on RPC requests for which there is no trustworthy authentication. In-transit traffic is protected, but the RPC client itself can still misrepresent user identity without server detection. TLS without authentication is an improvement from AUTH_SYS without encryption, but it leaves a critical security exposure.

In light of the above, when AUTH_SYS is used, the use of a TLS mutual authentication mechanism is **RECOMMENDED** to prove that the RPC client is known to the RPC server. The server can then determine whether the UIDs and GIDs in AUTH_SYS requests from that client can be accepted, based on the authenticated identity of the client.

The use of TLS does not enable RPC clients to detect compromise that leads to the impersonation of RPC users. Also, there continues to be a requirement that the mapping of 32-bit user and group ID values to user identities is the same on both the RPC client and server.

6.4. Best Security Policy Practices

RPC-with-TLS implementations and deployments are strongly encouraged to adhere to the following policies to achieve the strongest possible security with RPC-with-TLS.

- When using AUTH_NULL or AUTH_SYS, both peers are **RECOMMENDED** to have DNSSEC TLSA records, keys with which to perform mutual peer authentication using one of the methods described in Section 5.2, and a security policy that requires mutual peer authentication and rejection of a connection when host authentication fails.
- RPCSEC_GSS provides integrity and privacy services that are largely redundant when TLS is in use. These services **SHOULD** be disabled in that case.

7. IANA Considerations

7.1. RPC Authentication Flavor

Following Appendix B of [RFC5531], an entry has been added to the "RPC Authentication Flavor Numbers" registry. The purpose of the new authentication flavor is to signal the use of TLS with RPC. This new flavor is not a pseudo-flavor.

The fields in the new entry have been assigned as follows:

Identifier String: AUTH_TLS

Flavor Name: TLS

Value: 7

Description: Indicates support for RPC-with-TLS

Reference: RFC 9289

7.2. ALPN Identifier for SunRPC

Following Section 6 of [RFC7301], the following value has been allocated in the "TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry. The "sunrpc" string identifies SunRPC when used over TLS.

Protocol: SunRPC

Identification Sequence: 0x73 0x75 0x6e 0x72 0x70 0x63 ("sunrpc")

Reference: RFC 9289

7.3. Object Identifier for PKIX Extended Key Usage

Per the Specification Required policy defined in Section 4.6 of [RFC8126], the following new values have been registered in the "SMI Security for PKIX Extended Key Purpose" registry (1.3.6.1.5.5.7.3) (see Section 5.2.1.1 and Appendix B).

Decimal	Description	Reference
33	id-kp-rpcTLSClient	RFC 9289
34	id-kp-rpcTLSServer	RFC 9289

Table 1

7.4. Object Identifier for ASN.1 Module

Per the Specification Required policy defined in Section 4.6 of [RFC8126], the following new value has been registered in the "SMI Security for PKIX Module Identifier" registry (1.3.6.1.5.5.7.0) (see Appendix B).

Decimal	Description	Reference
105	id-mod-rpcWithTLS-2021	RFC 9289

Table 2

8. References

8.1. Normative References

[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>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <https://www.rfc-editor.org/info/rfc5056>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, DOI 10.17487/RFC5531, May 2009, <https://www.rfc-editor.org/info/rfc5531>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[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/info/rfc8446>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[RFC9266] Whited, S., "Channel Bindings for TLS 1.3", RFC 9266, DOI 10.17487/RFC9266, July 2022, <https://www.rfc-editor.org/info/rfc9266>.

[X.509] International Telecommunication Union, "Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks", ISO/IEC 9594-8, ITU-T Recommendation X.509, October 2019.

[X.680] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

[X.690] ITU-T, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, February 2021, <https://www.itu.int/rec/T-REC-X.690>.

8.2. Informative References

[RFC1833] Srinivasan, R., "Binding Protocols for ONC RPC Version 2", RFC 1833, DOI 10.17487/RFC1833, August 1995, <https://www.rfc-editor.org/info/rfc1833>.

[RFC2203] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, DOI 10.17487/RFC2203, September 1997, <https://www.rfc-editor.org/info/rfc2203>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <https://www.rfc-editor.org/info/rfc6698>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <https://www.rfc-editor.org/info/rfc7435>.

[RFC8166] Lever, C., Ed., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call Version 1", RFC 8166, DOI 10.17487/RFC8166, June 2017, <https://www.rfc-editor.org/info/rfc8166>.

[RFC8167] Lever, C., "Bidirectional Remote Procedure Call on RPC-over-RDMA Transports", RFC 8167, DOI 10.17487/RFC8167, June 2017, <https://www.rfc-editor.org/info/rfc8167>.

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

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

Appendix A. Known Weaknesses of the AUTH_SYS Authentication Flavor

The ONC RPC protocol, as specified in [RFC5531], provides several modes of security, commonly referred to as "authentication flavors". Some of these flavors provide much more than an authentication service. We refer to these as authentication flavors, security flavors, or simply, flavors. One of the earliest and most basic flavors is AUTH_SYS, also known as AUTH_UNIX. Appendix A of [RFC5531] specifies AUTH_SYS.

AUTH_SYS assumes that the RPC client and server both use POSIX-style user and group identifiers (each user and group can be distinctly represented as a 32-bit unsigned integer). It also assumes that the client and server both use the same mapping of user and group to an integer. One user ID, one primary group ID, and up to 16 supplemental group IDs are associated with each RPC request. The combination of these identifies the entity on the client that is making the request.

A string identifies peers (hosts) in each RPC request. [RFC5531] does not specify any requirements for this string other than that it is no longer than 255 octets. It does not have to be the same from request to request. Also, it does not have to match the DNS hostname of the sending host. For these reasons, even though most implementations fill in their hostname in this field, receivers typically ignore its content.

Appendix A of [RFC5531] contains a brief explanation of security considerations:

> It should be noted that use of this flavor of authentication does not guarantee any security for the users or providers of a service, in itself. The authentication provided by this scheme can be considered legitimate only when applications using this scheme and the network can be secured externally, and privileged transport addresses are used for the communicating end-points (an example of this is the use of privileged TCP/UDP ports in UNIX systems -- note that not all systems enforce privileged transport address mechanisms).

It should be clear, therefore, that AUTH_SYS by itself (i.e., without strong client authentication) offers little to no communication security:

1. It does not protect the confidentiality or integrity of RPC requests, users, or payloads, relying instead on "external" security.
2. It does not provide authentication of RPC peer machines, other than inclusion of an unprotected domain name.
3. The use of 32-bit unsigned integers as user and group identifiers is problematic because these data types are not cryptographically signed or otherwise verified by any authority. In addition, the mapping of these integers to users and groups has to be consistent amongst a server and its cohort of clients.
4. Because the user and group ID fields are not integrity protected, AUTH_SYS does not provide non-repudiation.

Appendix B. ASN.1 Module

The following module adheres to ASN.1 specifications [X.680] and [X.690].

```
<CODE BEGINS>


RPCwithTLS-2021
  { iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-rpcWithTLS-2021(105) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- OID Arc

id-kp OBJECT IDENTIFIER ::=
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) kp(3) }

-- Extended Key Usage Values

id-kp-rpcTLSClient OBJECT IDENTIFIER ::= { id-kp 33 }
id-kp-rpcTLSServer OBJECT IDENTIFIER ::= { id-kp 34 }

END


<CODE ENDS>
```

Acknowledgments

Special mention goes to Charles Fisher, author of "Encrypting NFSv4 with Stunnel TLS". His article inspired the mechanism described in the current document.

Many thanks to Benjamin Coddington, Tigran Mkrtchyan, and Rick Macklem for their work on prototype implementations and feedback on the current document. Also, thanks to Benjamin Kaduk for his expert guidance on the use of PKIX and TLS and to Russ Housley for his ASN.1 expertise and for providing other proper finishing touches. In addition, the authors thank the other members of the IESG for their astute review comments. These contributors made this a significantly better document.

Thanks to Derrell Piper for numerous suggestions that improved both this simple mechanism and the current document's security-related discussion.

Many thanks to Transport Area Director Magnus Westerlund for his sharp questions and careful reading of the final revisions of the current document. The text of Section 5.1.2 is mostly his contribution.

The authors are additionally grateful to Bill Baker, David Black, Alan DeKok, Lars Eggert, Olga Kornievskaia, Greg Marsden, Alex McDonald, Justin Mazzola Paluska, Tom Talpey, Martin Thomson, and Nico Williams for their input and support of this work.

Finally, special thanks to NFSV4 Working Group Chair and document shepherd David Noveck, NFSV4 Working Group Chairs Spencer Shepler and Brian Pawlowski, and NFSV4 Working Group Secretary Thomas Haynes for their guidance and oversight.

Authors' Addresses

Trond Myklebust
Hammerspace Inc.
4300 El Camino Real, Suite 105
Los Altos, CA 94022
United States of America
Email: trond.myklebust@hammerspace.com

Charles Lever (EDITOR)
Oracle Corporation
United States of America
Email: chuck.lever@oracle.com