Workgroup: Network File System Version 4
Internet-Draft: draft-ietf-nfsv4-rpc-tls-08
Updates: 5531 (if approved)
Published: 19 June 2020
Intended Status: Standards Track
Expires: 21 December 2020
Authors: T. Myklebust   C. Lever, Ed.
Hammerspace   Oracle

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 in-transit Remote Procedure Call (RPC) transactions while interoperating with ONC RPC implementations that do not support this mechanism. This document updates RFC 5531.

Discussion of this draft takes place on the NFSv4 working group mailing list (nfsv4@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/nfsv4/. Working Group information can be found at https://datatracker.ietf.org/wg/nfsv4/about/.

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

The source for this draft is maintained in GitHub. Suggested changes should be submitted as pull requests at https://github.com/chucklever/i-d-rpc-tls. Instructions are on that page as well.

Status of This Memo

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 is done by 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:

* Parts of each RPC header remain in clear-text, constituting a significant security exposure.

* Offloading the 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 UIDs and GIDs from unauthenticated clients brings with it. Continued use is in part because:

* Per-client deployment and administrative costs are not scalable. Administrators must provide keying material for each RPC client, including transient clients.

* Host identity management and user identity management must be enforced in the same security realm. In certain environments, different authorities might be responsible for provisioning client systems versus provisioning new users.

The alternative described in the current document is to employ a transport layer security mechanism that can protect the confidentiality of each RPC connection transparently to RPC and upper-layer protocols. The Transport Layer Security protocol [RFC8446] (TLS) is a well-established Internet building block that protects many standard Internet protocols such as the Hypertext Transport Protocol (HTTP) [RFC2818].

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, generating additional keying material, or provisioning a secure network tunnel.

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.

The current document specifies the implementation of RPC on an encrypted transport in a manner that is transparent to upper-layer protocols based on RPC. The imposition of encryption at the transport layer protects any upper-layer protocol that employs RPC, without alteration of that protocol.

Further, Section 7 of the current document defines policies in line with [RFC7435] which enable RPC-over-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 is used for all RPC transactions. Enforcing the use of RPC-over-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 RPC, TLS, PKI, GSS-API, and DNSSEC is already available in an RPC implementation where 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 Remote Procedure Call (RPC) version 2 protocol [RFC5531] and the Transport Layer Security (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-Over-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-over-TLS. Policy settings on the RPC-over-TLS-enabled peer determine whether RPC operation continues without the use of TLS or RPC operation is not permitted.

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 8.1.

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. 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 mechanism described in the current document does not support RPC transports other than TCP and UDP.

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.

If the RPC server replies 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 can continue, however it will be without any confidentiality, integrity or 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 setting the reply_stat field to MSG_DENIED, the reject_stat field to AUTH_ERROR, and the auth_stat field to AUTH_BADCRED.

Once the TLS session handshake is complete, the RPC client and server have established a secure channel for communicating. A successful AUTH_TLS probe on one particular port/transport tuple never implies RPC-over-TLS is available on that same server using a different port/transport tuple.

4.2. Authentication

Both RPC and TLS have peer and user authentication, with some overlap in capability between 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-over-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-over-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 (e.g. a certificate) that is backed by a trusted entity. 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.

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 must not utilize the remote TLS peer identity for RPC user authentication.

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 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 redundant. An RPC implementation capable of concurrently using TLS and RPCSEC GSS can use GSS channel binding, as defined in [RFC5056], to determine when an underlying transport provides a sufficient degree of confidentiality. Channel bindings for the TLS channel type are defined in [RFC5929].

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 v1.3 (for TLS [RFC8446] or DTLS [I-D.ietf-tls-dtls13] respectively). Support for mandatory-to-implement ciphersuites for the negotiated TLS version is **REQUIRED**.

* Implementations **MUST** support certificate-based mutual authentication. Support for TLS-PSK mutual authentication [RFC4279] is **OPTIONAL**. See Section 4.2 for further details.

* Negotiation of a ciphersuite providing confidentiality as well as integrity protection is **REQUIRED**. Support for and negotiation of compression is **OPTIONAL**.

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 8.2 in that message's ProtocolNameList value.

Similary, 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 8.2 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 traditionally a strong association between an RPC program and a 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 Transport Layer Security (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-over-TLS support for that connection. If spurious traffic appears on a TCP connection between the initial clear-text 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 clear-text 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 [RFC8166]). 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 and may then close the TCP connection.

5.1.2. Protected Operation on UDP

RFC Editor: In the following section, please replace TBD with the connection_id extension number that is to be assigned in [I-D.ietf-tls-dtls-connection-id]. And, please remove this Editor's Note before this document is published.

RPC over UDP is protected using the Datagram Transport Layer Security (DTLS) protocol [I-D.ietf-tls-dtls13].

Using DTLS does not introduce reliable or in-order semantics to RPC on UDP. Each RPC message **MUST** fit in a single DTLS record. DTLS encapsulation has overhead, which reduces the effective Path MTU (PMTU) and thus the maximum RPC payload size. The use of DTLS record replay protection is **REQUIRED** when transporting RPC traffic.

As soon as a client initializes a UDP socket for use with an RPC server, it uses the mechanism described in Section 4.1 to discover DTLS support for an RPC program on a particular port. It then negotiates a DTLS session.

Multi-homed 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(TBD)" extension described in Section 9 of [I-D.ietf-tls-dtls13], and RPC-on-DTLS peer endpoints **MUST** provide a ConnectionID with a non-zero 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. Subsequent RPC messages exchanged between the RPC client and server are no longer protected until a new DTLS session is established.

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

Implementations are **REQUIRED** to support this mechanism. In this mode, the tuple (serial number of the presented certificate; Issuer) uniquely identifies the RPC peer.

X.509 certificates are specified in [X.509] and extended in [RFC5280].

* Implementations **MUST** allow the configuration of a list of trusted Certification Authorities for authorizing incoming connections.

* Certificate validation **MUST** include the verification rules as per [RFC5280].

* Implementations **SHOULD** indicate their trusted Certification Authorities (CAs).

* Peer validation always includes a check on whether the locally configured expected DNS name or IP address of the server that is contacted matches its presented certificate. DNS names and IP addresses can be contained in the Common Name (CN) or subjectAltName entries. For verification, only one of these entries is to be considered. The following precedence applies: for DNS name validation, subjectAltName:DNS has precedence over CN; for IP address validation, subjectAltName:iPAddress has precedence over CN. Implementors of this specification are advised to read Section 6 of [RFC6125] for more details on DNS name validation.

* For services accessed by their network identifiers (netids) and universal network addresses (uaddr), the iPAddress subjectAltName **SHOULD** be present in the certificate and must exactly match the address represented by the universal network address.

* 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 or a set of allowed X.509v3 Certificate Policies).

* When the configured trust base changes (e.g., removal of a CA from the list of trusted CAs; issuance of a new CRL for a given CA), implementations **MAY** renegotiate the TLS session to reassess the connecting peer's continued authorization.

Authenticating a connecting entity does not mean the RPC server necessarily wants to communicate with that client. For example, if the Issuer is not in a trusted set of Issuers, the RPC server may decline to perform RPC transactions with this client.

Implementations that want to support a wide variety of trust models should expose as many details of the presented certificate to the administrator as possible so that the administrator can implement the trust model. As a suggestion, at least the following parameters of the X.509 client certificate **SHOULD** be exposed:

* Originating IP address

* Certificate Fingerprint

* Issuer

* Subject

* all X.509v3 Extended Key Usage

* all X.509v3 Subject Alternative Name

* all X.509v3 Certificate Policies

5.2.2. X.509 Certificates Using Fingerprints

This mechanism is **OPTIONAL** to implement. In this mode, the fingerprint of a certificate uniquely identifies the RPC peer.

A "fingerprint" is typically defined as a cryptographic digest of the Distinguished Encoding Rules (DER) form [X.690] of an X.509v3 certificate [X.509]. Implementations **SHOULD** allow the configuration of a list of trusted certificates that is indexed by fingerprint.

5.2.3. Pre-Shared Keys

This mechanism is **OPTIONAL** to implement. In this mode, the RPC peer is uniquely identified by keying material that has been shared out-of-band or by a previous TLS-protected connection (see Section 2.2 of [RFC8446]). At least the following parameters of the TLS connection **SHOULD** be exposed:

* Originating IP address

* TLS Identifier

5.2.4. Token Binding

This mechanism is **OPTIONAL** to implement. In this mode, a token uniquely identifies the RPC peer.

Versions of TLS after TLS 1.2 contain a token binding mechanism that is more secure than using certificates. This mechanism is detailed in [RFC8471].

6. Implementation Status

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

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs.

Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

6.1. DESY NFS server

Organization: DESY

URL: https://desy.de

Maturity: Implementation will be based on mature versions of the current document.

Coverage: The bulk of this specification is implemented including DTLS.

Licensing: LGPL

Implementation experience: The implementer has read and commented on the current document.

6.2. Hammerspace NFS server

Organization: Hammerspace

URL: https://hammerspace.com

Maturity: Prototype software based on early versions of the current document.

Coverage: The bulk of this specification is implemented. The use of DTLS functionality is not implemented.

Licensing: Proprietary

Implementation experience:
No comments from implementors.

6.3. Linux NFS server and client

Organization: The Linux Foundation

URL: https://www.kernel.org

Maturity: Prototype software based on early versions of the current document.

Coverage: The bulk of this specification has yet to be implemented. The use of DTLS functionality is not planned.

Licensing: GPLv2

Implementation experience: No comments from the implementor.

6.4. FreeBSD NFS server and client

Organization: The FreeBSD Project

URL: https://www.freebsd.org

Maturity: Prototype software based on early versions of the current document.

Coverage: The bulk of this specification is implemented. The use of DTLS functionality is not planned.

Licensing: BSD

Implementation experience: Implementers have read and commented on the current document.

7. 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. Implementers should familiarize themselves with these materials.

7.1. Limitations of an Opportunistic Approach

The purpose of using an explicitly opportunistic approach is to enable interoperation with implementations that do not support RPC-over-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-over-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.

7.1.1. STRIPTLS Attacks

A classic form of attack on network protocols that initiate an association in plain-text to discover support for TLS is a man-in-the-middle that alters the plain-text handshake to make it appear as though TLS support is not available on one or both peers. Clients 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 hostname to X.509 identity. If TLS cannot be negotiated or authentication fails, the client disconnects and reports the problem.

* 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. If TLSA records are not available, this approach is strongly encouraged.

7.1.2. 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 clear-text 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 connnection handshake information and TLS certificate material that might enable attackers 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.

7.2. TLS Identity Management on Clients

The goal of the RPC-over-TLS protocol extension is to hide the content of RPC requests while they are in transit. The RPC-over-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 identity material is 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 identity material so that local access to it is restricted to only the local users that are authorized to perform operations on the remote RPC server.

7.3. 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 non-participant messages. In particular, transport layer encryption plus peer authentication protects receiving XDR decoders from deserializing untrusted data, a common coding vulnerability.

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, it is **RECOMMENDED** that when AUTH_SYS is used, every RPC client should present host authentication material to RPC servers to prove that the client is a known one. The server can then determine whether the UIDs and GIDs in AUTH_SYS requests from that client can be accepted.

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.

7.4. Best Security Policy Practices

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

* When using AUTH_NULL or AUTH_SYS, both peers are required to have DNS TLSA records and certificate material, and a policy that requires mutual peer authentication and rejection of a connection when host authentication fails.

* RCPSEC_GSS provides integrity and privacy services which are redundant when TLS is in use. These services should be disabled in that case.

8. IANA Considerations

RFC Editor: In the following subsections, please replace RFC-TBD with the RFC number assigned to this document. And, please remove this Editor's Note before this document is published.

8.1. RPC Authentication Flavor

Following Appendix B of [RFC5531], the authors request a single new entry in 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 are assigned as follows:

Identifier String: AUTH_TLS

Flavor Name: TLS

Value:
7

Description: Indicates support for RPC-over-TLS.

Reference: RFC-TBD

8.2. ALPN Identifier for SUNRPC

Following Section 6 of [RFC7301], the authors request the allocation of the following value in the "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-TBD

9. References

9.1. Normative References

[I-D.ietf-tls-dtls-connection-id]
Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", Work in Progress, Internet-Draft, draft-ietf-tls-dtls-connection-id-07, 21 October 2019, <https://tools.ietf.org/html/draft-ietf-tls-dtls-connection-id-07>.

[I-D.ietf-tls-dtls13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-38, 29 May 2020, <https://tools.ietf.org/html/draft-ietf-tls-dtls13-38>.

[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>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <https://www.rfc-editor.org/info/rfc4279>.

[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>.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <https://www.rfc-editor.org/info/rfc5929>.

[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>.

[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>.

[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>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[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>.

[X.509] International Telephone and Telegraph Consultative Committee, "ITU-T X.509 - Information technology - The Directory: Public-key and attribute certificate frameworks.", ISO/IEC 9594-8, CCITT Recommendation X.509, October 2019.

[X.690]
International Telephone and Telegraph Consultative Committee, "ITU-T X.690 - Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ISO/IEC 8825-1, CCITT Recommendation X.690, August 2015.

9.2. Informative References

[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>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[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>.

[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>.

[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>.

[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>.

[RFC8471] Popov, A., Ed., Nystroem, M., Balfanz, D., and J. Hodges, "The Token Binding Protocol Version 1.0", RFC 8471, DOI 10.17487/RFC8471, October 2018, <https://www.rfc-editor.org/info/rfc8471>.

Appendix A. Known Weaknesses of the AUTH_SYS Authentication Flavor

The ONC RPC protocol, as specified in [RFC5531], provides several modes of security, traditionally 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 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.

4. Because the user and group ID fields are not integrity-protected, AUTH_SYS does not provide non-repudiation.

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 Tigran Mkrtchyan and Rick Macklem for their work on prototype implementations and feedback on the current 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, Benjamin Kaduk, Olga Kornievskaia, Greg Marsden, Alex McDonald, Justin Mazzola Paluska, Tom Talpey, and Martin Thomson 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 Ste 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