Network File System Version 4
Internet-Draft
Updates: 5531 (if approved)
Intended status: Standards Track
Expires: May 23, 2019
November 19, 2018

T. Myklebust
Hammerspace
C. Lever, Ed.
Oracle

Remote Procedure Call Encryption By Default
draft-cel-nfsv4-rpc-tls-01

Abstract

This document describes a mechanism that enables encryption of in-transit Remote Procedure Call (RPC) transactions with little administrative overhead and full interoperation with RPC implementations that do not support this mechanism. This document updates RFC 5531.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In 2014 the IETF published [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 make a real effort to mitigate monitoring attacks. Typically this mitigation is done by encrypting data in transit.

The Remote Procedure Call version 2 protocol has been around for three decades (see [RFC5531] and its antecedants). Eisler et al. first introduced an in-transit encryption mechanism for RPC with RPCSEC GSS years ago [RFC2203]. However, experience has shown that RPCSEC GSS is challenging to deploy, especially in environments where:

- Per-host administrative or deployment costs must be kept to a minimum,

- Parts of the RPC header that remain in clear-text are a security exposure,

- Host CPU resources are at a premium, or

- Host identity management is carried out in a security domain that is distinct from user identity management.

However strong a privacy service is, it is not effective if it cannot be deployed in typical environments.

An alternative approach is to employ a transport layer security mechanism that can protect the privacy 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 common Internet protocols such as the Hypertext Transport Protocol (http) [RFC2818].

Encrypting at the RPC transport layer enables several significant benefits.

Encryption By Default
Via the use of self-signed certificates, in-transit encryption can be enabled immediately after installation without additional administrative actions such as identifying the host system to a trust authority, generating additional key material, or provisioning a secure network tunnel.

Protection of Existing Protocols
The imposition of encryption at the transport layer protects any Upper Layer protocol that employs RPC without alteration of that protocol. RPC transport layer encryption can protect recent versions of NFS such as NFS version 4.2 [RFC7862] and indeed legacy NFS versions such as NFS version 3 [RFC1813] and NFS side-band protocols such as the MNT protocol [RFC1813].

Decoupled User and Host Identities
RPCSEC GSS provides a framework for cryptographically protecting user and host identities but assumes that both are managed by the same security authority.

Encryption Offload
The use of a well-established transport encryption mechanism that is also employed by other very common network protocols makes it possible to use hardware encryption implementations so that the

host CPU is not burdened with the work of encrypting and decrypting large RPC arguments and results.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [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 uses the term "privacy" where other Internet communities might use "confidentiality". In this document the two terms are synonymous.

4. RPC-Over-TLS in Operation

4.1. Discovering Server-side TLS Support

The mechanism described in this document interoperates fully with implementations that do not support it. The use of TLS is automatically disabled in these cases. To achieve this, we introduce a new authentication flavor called AUTH_TLS. This new flavor is used to signal that the client wants to initiate TLS negotiation if the server supports it.

```
<CODE BEGINS>

enum auth_flavor {
        AUTH_NONE       = 0,
        AUTH_SYS        = 1,
        AUTH_SHORT      = 2,
        AUTH_DH         = 3,
        AUTH_KERB       = 4,
        AUTH_RSA        = 5,
        RPCSEC_GSS      = 6,
        AUTH_TLS        = 7,

        /* and more to be defined */
};

<CODE ENDS>
```

The length of the opaque data constituting the credential sent in the 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 received in the reply message from the server MUST be AUTH_NONE. The bytes of the verifier's string encode the fixed ASCII characters "STARTTLS".

When an RPC client is ready to initiate a TLS handshake, it sends a NULL RPC request with an auth_flavor of AUTH_TLS. The NULL request is made to the same port as if TLS were not in use.

The RPC server can respond in one of three ways:

- If the RPC server does not recognise the AUTH_TLS authentication flavor, it responds with a reject_stat of AUTH_ERROR. The RPC client then knows that this server does not support TLS.

- If the RPC server accepts the NULL RPC procedure, but fails to return an AUTH_NONE verifier containing the string "STARTTLS", the RPC client knows that this server does not support TLS.

- If the RPC server accepts the NULL RPC procedure, and returns an AUTH_NONE verifier containing the string "STARTTLS", the RPC client MAY proceed with TLS negotiation.

If an RPC client attempts to use AUTH_TLS for anything other than the NULL RPC procedure, the RPC server responds with a reject_stat of AUTH_ERROR.

Once the TLS handshake is complete, the RPC client and server will have established a secure channel for communicating and can proceed to use standard security flavors within that channel, presumably after negotiating down the irrelevant RPCSEC_GSS privacy and integrity services and applying channel binding [RFC7861].

If TLS negotiation fails for any reason -- say, the RPC server rejects the certificate presented by the RPC client, or the RPC client fails to authenticate the RPC server -- the RPC client reports this failure to the calling application the same way it would report an AUTH_ERROR rejection from the RPC server.

4.2. Streams and Datagrams

RPC commonly operates on stream transports and datagram transports. When operating on a stream transport, using TLS [RFC8446] is appropriate. On a datagram transport, RPC can use DTLS [RFC6347].

RPC-over-RDMA [RFC8166] may make use of transport layer security below the RDMA transport layer.

4.3. Authentication

Both RPC and TLS have their own forms of host and user authentication. We believe the combination of host authentication via TLS and user authentication via RPC provides optimal security, efficiency, and flexibility, although many combinations are possible.

TLS encryption-only with AUTH_SYS: A self-signed certificate enables TLS encryption. The RPC client uses AUTH_SYS to identify users with the guarantee that the UID and GID values cannot be observed or altered in transit. End-to-end encryption is provided via per-client certificate material that can be generated automatically.

TLS per-client certificate with AUTH_SYS: During TLS negotiation, the client identifies itself to the server with a unique certificate. As with encryption-only with AUTH_SYS, UID and GID values are well protected. In addition, the server can use the client's identity to perform additional authorization of this client's requests.

TLS encryption-only with RPCSEC GSS Kerberos: A self-signed certificate enables TLS encryption in encryption-only mode. The RPC client uses Kerberos to identify the client host and its users, and therefore does not need to enable costly GSS integrity or privacy services.

TLS per-user certificate with AUTH_NONE: Each user establishes her own TLS context with the server, identified by a unique certficate. There is no need for any additional information at the RPC layer, so the RPC client can use the simplest authentication flavor for RPC transactions. This configuration is not typical for NFS deployments, but it does enable strong certificate-based user authentication, which is currently not afforded by GSS.

[This is currently the most skeletal section in the document. There are two key areas for improvement:

o The interoperability of this new security flavor likely depends on us culling the "many combinations" down to just a few. At least we need to identify which ones are least workable, and provide some operational details for the important commonly deployed cases.

o There might be opportunity to re-examine the efficacy of host authentication. If host authentication does not provide significant increases in security, perhaps we can get away with specifying only encryption-only configurations.

-Ed.]

5. Security Considerations

One purpose of the mechanism described in this document is to protect RPC-based applications against threats to the privacy of RPC transactions and RPC user identities. A taxonomy of these threats appears in Section 5 of [RFC6973]. In addition, Section 6 of [RFC7525] contains a detailed discussion of technologies used in conjunction with TLS. Implementers should familiarize themselves with these materials.

The NFS version 4 protocol permits more than one user to use an NFS client at the same time [RFC7862]. Typically that NFS client will conserve connection resources by routing RPC transactions from all of its users over a few or a single connection. In circumstances where the users on that NFS client belong to multiple distinct security domains, it may be necessary to establish separate TLS-protected connections that do not share the same encryption parameters.

6. IANA Considerations

This document does not require actions by IANA.

7. References

7.1. 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>.

[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>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[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>.

[RFC7861] Adamson, A. and N. Williams, "Remote Procedure Call (RPC) Security Version 3", RFC 7861, DOI 10.17487/RFC7861, November 2016, <https://www.rfc-editor.org/info/rfc7861>.

[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>.

[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>.

7.2. Informative References

[LJNL] Fisher, C., "Encrypting NFSv4 with Stunnel TLS", August 2018, <https://www.linuxjournal.com/content/encrypting-nfsv4-stunnel-tls>.

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, DOI 10.17487/RFC1813, June 1995, <https://www.rfc-editor.org/info/rfc1813>.

[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>.

[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>.

[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>.

[RFC7862] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 Protocol", RFC 7862, DOI 10.17487/RFC7862, November 2016, <https://www.rfc-editor.org/info/rfc7862>.

Acknowledgments

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

The authors are grateful to Bill Baker, David Black, Benjamin Kaduk Greg Marsden, David Noveck, Justin Mazzola Paluska, and Tom Talpey for their input and support of this work.

Special thanks go to Transport Area Director Spencer Dawkins, 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
1015 Granger Avenue
Ann Arbor, MI 48104
United States of America

Email: chuck.lever@oracle.com