Linux kernel RPC-with-TLS

Implementation Status, Late 2021

Chuck Lever – October 5, 2021

What Is RPC-with-TLS?

- TLS here means specifically Transport Layer Security v1.3 [RFC 8446]
- RPC-with-TLS enables a new RPC security mode
  - Encryption of whole RPC messages
  - Hardware offload supported
  - Peer authentication using either X.509 certificates or pre-shared keys
  - Existing RPC security flavors provide user identification/authentication

What Is RPC-with-TLS?

- Many of the benefits of RPCSEC GSS without the hassles of Kerberos
  - The protocol does not require the use of client certificates
  - If only in-transit encryption is desired, the server needs a self-signed certificate, and that’s it
  - Client authentication requires a common CA to provide certificates to clients and the server

What Is RPC-with-TLS?

- Opportunistic use of TLS
  - RPC-with-TLS service is available on the same destination port as non-TLS RPC
  - Security policies are left up to local administrators
    - Backwards-compatible: Use TLS only when both sides can
    - Best security: Enforce the use of TLS to thwart STRIPTLS attacks

TLSv1 Basics

- Provides peer authentication and traffic encryption
- Used by many popular Internet protocols: HTTPS, DNS, SMTP, NVMe/TCP
- Two subprotocols
  - TLS handshake protocols - authenticate peers, then negotiate crypto algorithm and key
  - TLS record protocol - given algorithm choice and session key, handle data encryption and decryption

Current Linux Kernel TLS Implementation

- AFAIK there are no in-kernel TLS consumers today
- Kernel already implements the TLS record protocol (kTLS)
  - Both hardware offload and software only
- Kernel does not implement any TLS handshake protocol
  - Existing TLS consumers pass a session key negotiated by user space libraries to the kernel via ioctl

Possible Kernel TLS Handshake Designs

- Accept connections in user space and perform handshake there, then pass accepted secure socket endpoint to kernel
- Accept connections in kernel, hand socket endpoint to user space to perform handshake which returns session key
- Hand certificate material to kernel and let it deal with accepting each connection and performing the TLS handshake

Tempesta TLS Design

https://github.com/tempesta-tech/tempesta

- Based on tls.mbed.org, now trustedfirmware.org
- User space places X.509 certificates on keyrings
  - Both EE, intermediate, and CA certs are provided to the kernel
  - Tempesta kernel module uses these for handshake protocol
- Existing kernel X.509 and crypto would assist the handshake, then existing kTLS infrastructure would handle TLS record protocol

Tempesta To-Do

- Extract the handshake code from Tempesta FW product, convert it into a loadable kernel module
- Implement a ClientHello API to go with ServerHello API
- Update TLS v1.2 (RFC 5246) support to TLS v1.3 (RFC 8446)

RPC-with-TLS Design

- RPC-with-TLS allows “opportunistic” use of TLS
  - A special RPC probe is done by clients to detect when the server can do RPC-with-TLS
  - If server support is detected, clients initiate a TLSv1.3 handshake
  - Subsequently all RPC traffic to that port is wrapped in the established TLS session

Linux NFS Client Goals

- NFS mount option to control when TLS will be used
  - Choose: Always on / only when server has TLS / or always off
- Support for full mutual peer authentication, server-only, or none
  - mount.nfs to inject correct certificates into kernel based on net namespace and source and destination IP addresses

RPC-with-TLS Client Progress

- auth_tls.c implemented
- RPC FSM modified to emit RPC_AUTH_TLS probe when needed
- NFS mount option: “tls=none|auto|required” (with man page updates)
- Only TCP is supported
- No Tempesta module support for ClientHello yet
- No client-side peer authentication yet

Linux NFS Server Goals

- Export/share options to control when TLS is required or optional
- TLS configuration to control how much client certificate validation to perform
- rpc.nfsd to inject local server certificate plus chain of trust into kernel based on net namespace and local IP address

RPC-with-TLS Server Progress

- svcauth_tls implemented
- Server can recognize and respond to RPC_AUTH_TLS probe
- Only TCP is supported
- Exploring existing ServerHello API in the Tempesta module
- No server-side user space support yet (exportfs or nfsd)

Supplemental Material

FIPS 140-2 (and -3) Compliance

- FIPS 140-2 is a US government information processing standard that mandates the use of a narrow set of crypto algorithms and other requirements
  - Compliance of an OS implementation is determined by regular review and extensive testing
  - FIPS compliance is frequently a requirement for government information services that run in the cloud
- Since the Tempesta in-kernel TLS handshake code will be modified to use existing kernel X.509 and crypto infrastructure which has already been reviewed and tested for FIPS compliance, it should not be difficult for TLS to pass subsequent review

TLS User Squashing

- RPC-with-TLS itself provides no innate form of user identification
- FreeBSD NFS servers check for special fields in client certificates that contain a user@domain value
- The server then squashes all RPC user identities on that TLS session to the specified user
- If that field is not present, traditional RPC user identification is used

How Can QUIC Benefit From In-kernel TLS?

- The QUICv1 protocol uses the TLS v1.3 handshake protocol but has its own record protocol
- The Tempesta in-kernel handshake code could be re-used for an in-kernel QUIC implementation

Bibliography

- https://www.rfc-editor.org/info/rfc5246
- https://www.rfc-editor.org/info/rfc8446
- https://datatracker.ietf.org/doc/draft-ietf-nfsv4-rpc-tls/
- https://git.kernel.org/pub/scm/linux/kernel/git/cel/linux.git/log/?h=topic-rpc-with-tls
- https://github.com/chucklever/i-d-rpc-over-quicv1
- https://lwn.net/Articles/866481/