s2n(3) 0.6.0 | s2n Tcl wrapper

Cyan Ogilvie

S2N

s2n Tcl wrapper - layer TLS onto Tcl channels

SYNOPSIS

package require s2n ?0.6.0?

s2n::push *channelName* ?*-opt val ...*?
s2n::socket ?*-opt val ...*? *host port*

DESCRIPTION

This package provides a channel driver that can be stacked on any Tcl channel that supports reading and writing to add TLS support. The TLS implementation uses Amazon's s2n for the TLS implementation and aws-lc for the libcrypto implementation.

COMMANDS

s2n::push *channelName ?-opt val ...?* Stack a TLS protocol driver onto the channel *channelName*. See **OPTIONS** for the allowed options. The TLS protocol driver may be removed with the standard **chan pop** *channelName* Tcl command. The *channelName* channel may be blocking or non-blocking, and may be a socket created with **-async** and still be waiting for the connection to complete.

s2n::socket ?*-opt val ...*? *host port* Open a TCP connection to *host*:*port* and handshake with that host as a TLS client. See **OPTIONS** for the available options. If *host* is not a numeric address (IPv4 or IPv6) then it supplies the default for the **-servername** option. If *host* is given as an empty string, then *port* is taken to be a filesystem path and a connection is made to an AF_UNIX socket at that path.

OPTIONS

-config ***config*** Override the default configuration for the channel, see **CONFIG** for details.

-role client|server Set the role the TLS driver will play in the TLS handshake. If **client** (the default) the driver will initiate a TLS handshake otherwise it will wait to receive a ClientHello TLS message and handshake as a server. Only valid as an option to the **s2n::push** command, sockets opened using **s2n::socket** area always clients.

-servername ***host*** Set the SNI (Server Name Indication) name to send when handshaking as a client. There is no default, if not present, the SNI extension won't be used. It's also an error to set this option for server connections.

-prefer throughput|latency Tune the implementation to optimise for throughput (large frames, fewer syscalls) or latency (smaller frames, more syscalls). TODO: figure out the default.

-async Only valid for **s2n::socket**: don't block on establishing the connection to *host*:*port*, but return immediately. If data is written in blocking mode before the connection is established and the TLS handshake is completed then the write will block until these are done and the data is written. In non-blocking mode the channel will become writable when the TLS handshake completes, and readable once application data arrives from the peer.

CONFIG

The *config* dictionary can contain the following keys to override the default configuration for the connection. An error will be thrown if the dictionary contains any unrecognised keys.

session_tickets ***bool*** If set to a true boolean value, enable session tickets for this connection. Session tickets are a way to bootstrap future connections with a server without going through the full certificate-based key exchange, enabling lower latency connection establishment.

ticket_lifetime {*encrypt_decrypt_seconds decrypt_only_seconds*} Set the time for which session tickets are valid, as a list of two values. The first, *encrypt_decrypt_seconds* is the time for which the ticket may be used to both encrypt and decrypt. The second, *decrypt_only_seconds*, is the time during which the key cannot be used to encrypt but may still decrypt.

cipher_preferences ***policy*** Select the set of allowed ciphers and their preferences, via the *policy*, which is a security policy string as understood by s2n, like "default_tls13" or "20230317".

EXAMPLES

Connect to an HTTPS server, request /

```
set sock    [socket www.google.com 443]
s2n::push $sock -servername www.google.com
chan configure $sock -buffering none -translation crlf -encoding ascii
puts $sock "GET / HTTP/1.1\nHost: www.google.com\nConnection: close\n"
while {[set line [gets $sock]; string length $line]} {
    puts $line
}
puts ""
chan configure $sock -translation binary
puts [read $sock]
close $sock
```

BUILDING

This package has no external dependencies other than Tcl. The s2n and aws-lc libraries it depends on are included as submodules (or baked into the release tarball) and are built and statically linked as part of the package build process.

Currently Tcl 8.7 is required, but if needed polyfills could be built to support 8.6.

From a Release Tarball

Download and extract the release, then build in the standard TEA way:

```
wget https://github.com/cyanogilvie/tcl-s2n/releases/download/v0.6.0/tcl-s2n-0.6.0.tar.gz
tar xf tcl-s2n-0.6.0.tar.gz
cd tcl-s2n-0.6.0
./configure
make
sudo make install
```

From the Git Sources

Fetch the code and submodules recursively, then build in the standard autoconf / TEA way:

```
git clone --recurse-submodules https://github.com/cyanogilvie/tcl-s2n
cd tcl-s2n
autoconf
./configure
make
sudo make install
```

In a Docker Build

Build from a specified release version, avoiding layer pollution and only adding the installed package without documentation to the image, and strip debug symbols, minimising image size:

```
WORKDIR /tmp/tcl-s2n
RUN wget https://github.com/cyanogilvie/tcl-s2n/releases/download/v0.6.0/tcl-s2n-0.6.0.tar.g
    ./configure; make test install-binaries install-libraries && \
    strip /usr/local/lib/libs2n*.so && \
    cd .. && rm -rf tcl-s2n
```

For any of the build methods you may need to pass `--with-tcl /path/to/tcl/lib` to `configure` if your Tcl install is somewhere nonstandard.

Testing

Since this package deals with security sensitive code, it's a good idea to run the test suite after building (especially in any automated build or CI/CD pipeline):

```
make test
```

And maybe also the memory checker `valgrind` (requires that Tcl and this package are built with suitable memory debugging flags, like `CFLAGS="-DPURIFY -Og" --enable-symbols`):

```
make valgrind
```

SECURITY

Given the limitations of a scripting language environment, this package's code does not have sufficient control over freed memory contents (or memory paged to disk) to guarantee that key material or other sensitive material (like decrypted messages) can't leak in a way that could be exploited by other code running on the shared memory (or disk) machine. For this reason, careful consideration should be given to the security requirements of the application as a whole when using this package in a shared execution context, or in a virtual machine.

MLOCK CONSIDERATIONS

The s2n-tls library uses mlock by default to prevent memory containing key material from being paged to disk (where it could leak to other processes or be recovered by a later forensic analysis of the disk). Unfortunately containers default to removing the needed capability to allow that (IPC_LOCK). That manifests as an error thrown during s2n initialisation:

```
% docker run --rm -it cyanogilvie/alpine-tcl:v0.9.87-stripped
tclsh8.7 [/here] package require s2n
s2n_init failed: error calling mlock (Did you run prlimit?)
while evaluating package require s2n
```

```
tclsh8.7 [/here] set errorCode
S2N S2N_ERR_MLOCK
tclsh8.7 [/here]
```

If you need to use this package in a container and are able to modify the capabilities then granting the IPC_LOCK capability will permit s2n to use mlock. For example, using docker that looks like this:

```
% docker run --rm -it --cap-add IPC_LOCK cyanogilvie/alpine-tcl:v0.9.87-stripped
tclsh8.7 [/here] package require s2n
0.6.0
tclsh8.7 [/here]
```

If you cannot add the IPC_LOCK capability then s2n's use of mlock can be disabled by setting the S2N_DONT_MLOCK environment variable (at the cost of losing the mlock protection for key material):

```
% docker run --rm -it -e S2N_DONT_MLOCK=1 cyanogilvie/alpine-tcl:v0.9.87-stripped
tclsh8.7 [/here] package require s2n
0.6.0
tclsh8.7 [/here]
```

FUZZING

TODO

AVAILABLE IN

The most recent release of this package is available by default in the `alpine-tcl` container image: docker.io/cyanogilvie/alpine-tcl and the `cftcl` Tcl runtime snap: https://github.com/cyanogilvie/cftcl.

SEE ALSO

This package is built on the s2n library and the aws-lc library.

PROJECT STATUS

This is a very early work in progress, but is in limited production use.

With the nature of this package a lot of care is taken with memory handling and test coverage. There are no known memory leaks or errors, and the package is routinely tested by running its test suite (which aims at full coverage) through valgrind. The `make valgrind`, `make test` and `make coverage` build targets support these goals. The test suite is currently a long way from full coverage.

BUGS

There are currently problems with the **s2n::socket** command in **-async** mode, and handling of orphaned channels created by **s2n::socket** during process exit or unloading this extension's dll, for now it is recommended to use the stacked channel approach with **s2n::push** instead.

SOURCE CODE

This package's source code is available at https://github.com/cyanogilvie/tcl-s2n. Please create issues there for any bugs discovered.

LICENSE