License-compliant TLS stack for Apertis targets

Contents

The Apertis distribution provides both a development environment for electronic devices as well as a software stack to be used on them. In line with this goal, the Apertis project strives to provide software components that, where there is intent that they form part of the software stack on the devices themselves, are free from licensing constraints that may make it unsuitable in certain use cases. An example is software licensed under the terms of the GNU GPL-3[1] (General Public License) or LGPL-3[2] (Lesser General Public License) which are known to present a problem as they sometimes conflict with regulatory requirements[3] and thus Apertis will take measures to avoid such packages being provided as part of the "target"package repositories[4].

[1] https://www.gnu.org/licenses/gpl-3.0.en.html
[2] https://www.gnu.org/licenses/lgpl-3.0.en.html
[3] https://www.apertis.org/policies/license-expectations/#licensing-constraints
[4] https://www.apertis.org/policies/license-expectations/#apertis-repository-component-specific-rules

Goals and requirements

The goal here is to provide TLS functionality not just for the packages contained within its own repositories, but to support applications added by those utilizing Apertis as well.

- **Requirement:** TLS implementation does not require code covered by licenses that are incompatible with the target repositories rules
- **Requirement:** TLS implementation is licensed under terms that does not preclude its use from existing target applications
- **Requirement:** TLS implementation is licensed under terms that does not preclude its use from users proprietary applications

Given the security sensitive nature of the TLS stack, utilizing unmaintained software here would be best avoided. Putting maintenance aside, these versions of their respective TLS implementations may not be gaining support for any new ciphers and TLS protocol versions, which will severely limit their usefulness as time progresses. As well as not gaining newer protocol versions, the libraries may not be updated to reflect the frequently changing recommendations regarding minimal protocol versions[5] that should be supported, which may result in issues when attempting to access sites following the "Modern"recommendation. Additionally, it is likely that newer versions of the packages utilizing these TLS implementations will begin to require functionality added to newer versions of the TLS libraries thus reducing the ability of Apertis to upgrade to these too.

TLS stack

In order to have up to date libraries, specially TLS ones which very important for security reasons Apertis based them on Debian as covered in the Apertis Release Flow[6] which present the following issues for Apertis

GnuTLS

Whilst GnuTLS is licensed under the LGPL-2.1[7], it uses Nettle[8] and GMP[9]. Newer versions of both of these dependencies are now licensed as dual GPL-2 or LGPL-3, rather than **L**GPL-2.1.

To avoid including GnuTLS under LGPL-3 terms since it is against Apertis license expectations[10], Apertis would need to utilize it under the GPL-2 terms. This would result in the binary GnuTLS library effectively being used under the terms of the GPL-2 rather than LGPL-2.1. This would restrict Apertis

[5] https://wiki.mozilla.org/Security/Server_Side_TLS
[6] https://www.apertis.org/policies/release-flow/#apertis-release-flow
[7] https://www.gnu.org/licenses/old-licenses/lgpl-2.1.en.html
[8] https://www.lysator.liu.se/~nisse/nettle/nettle.html
[9] https://gmplib.org/
[10] https://www.apertis.org/policies/license-expectations/

users from using this Apertis provided TLS implementation either directly or indirectly from any non-GPL-2 compatible applications they wish to integrate into their systems, for example in proprietary applications, where it would have the effect of requiring the app to also be GPL-2 licensed.

In such a scenario, a newer GnuTLS library could be allowed by accepting its dependencies under the GPL-2 license and restricting its use to places where this license wouldn't be problematic, such as existing GPL-2 software. As the existing applications written exclusively to use GnuTLS are GPL-2 or tolerant of GPL-2, this is viable.

OpenSSL

The OpenSSL version 1.1 available in v2022 and v2023 is licensed under a custom GPL-incompatible license. OpenSSL 3.0 available from v2024dev2 is licensed under the Apache 2.0[11] license, which is compatible with the GPL-3, but not GPL-2. This means that GPL-2 tools like `systemd` cannot use the newer versions of OpenSSL without effectively becoming GPL-3 licensed or through these upstream projects applying a license exceptions (for example as OpenVPN[12] has).

Fortunately, the GPL states[13] "as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable". If the library is distributed as part of the OS and can be considered a major component of it, then this clause doesn't require the library to be considered as part of the software and therefore falls outside of the scope of the license. A counter argument to this is that because the application may also be considered to be distributed as part of the operating system this exception doesn't apply especially in embedded devices where the software is distributed preinstalled as a complete entity.

Currently, most distributions such as Fedora and Debian consider OpenSSL a system library overcoming the incompatibility.

In relation to proprietary code, OpenSSL 1.1 is licensed under OpenSSL license a BSD-style license with additional advertising clauses. This license falls under the permissive category since it does not imposes many restrictions. OpenSSL 3.0 is licensed under the standard Apache 2.0, which is also a permissive one. In conclusion, the use of OpenSSL is suitable in proprietary code.

NSS

Network Security Services[14] (NSS) is a set of security libraries developed by

[11] https://www.apache.org/licenses/LICENSE-2.0
[12] https://spdx.org/licenses/openvpn-openssl-exception.html
[13] https://www.gnu.org/licenses/old-licenses/gpl-2.0.html#section3
[14] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS

Mozilla. NSS provides its own API, which is currently only supported by a few of the applications which use TLS in Apertis. It is licensed as MPL-2.0[15].

Approach

In order to fullfil the requirements the approach taken has been to upgrade GnuTLS to a new version for those applications that can use it licensed as GPL-2. With OpenSSL upgraded and retained as a system library, utilizing it, inline with the approach taken by other distributions that have documented a specific policy covering this.

The one outlier is the printing support in GTK which uses GnuTLS and which potentially ends up causing GPL-2 dependencies in GTK. Whilst Debian have also declared CUPS as a system library, we feel that the differing use cases for Debian and Apertis make this less of a realistic position to take. We have therefore dropped printing support from GTK in order to remove this dependency as we don't feel that this functionality is critical to Apertis'aim.

This approach was introduced initially in v2022, and after being tested the changes were backported to v2021 and v2020, to make older releases comply with Apertis license expectations.

Summary

The tables below summarize the use of TLS libraries in various releases of Apertis target images. We would expect proprietary applications to either utilize the OpenSSL or NSS libraries as deemed appropriate by the individual projects.

TLS stack

Component	License	OpenSSL	GnuTLS	Notes
apt	GPL-2+		X	
connman	GPL-2		X	
curl	curl and BSD-3-Clause and BSD-4-Clause-UC and ISC	X	X	also produ
glib-networking	LGPL-2.1+ and LGPL-2.1+ with OpenSSL exception	X		after reba
liboauth	Expat/MIT	curl		
~~libmicrohttpd~~	LGPL-2.1+		X	removed s
neon27	LGPL-2.1+	X	X	
openjpeg	BSD-2	curl		package li
openldap	OLDAP-2.8	X		
~~rtmpdump~~	GPL-2+ (tools), LGPL-2.1+ (library)		X	removed s
systemd	LGPL-2.1+ and GPL-2[+] and PD X	curl		package sy

[15] https://www.mozilla.org/en-US/MPL/2.0/

Component	License	OpenSSL	GnuTLS	Notes
tumbler	LGPL-2.1+ and GPL-2+	curl		

Appendix

Previous situation

The "target"section of Apertis ships a variety of packages which use TLS from a provided library. There are a number of software libraries that provide competing TLS implementations and which are provided under various licensing terms. However, these projects do not always provide the same programming interfaces, thus do not provide a drop in replacement for each other. Whilst some users of TLS libraries may provide some level of abstraction to support more than one TLS library, others may support only one and thus Apertis currently provides GnuTLS[16], OpenSSL[17] and NSS[18].

- **GnuTLS**: Apertis currently provides GnuTLS version 3.4.10. This is an approximately four-year-old version of GnuTLS as shipped in Ubuntu Xenial and thus is currently supported by Ubuntu and is expected to be until 2022. GnuTLS is used directly or indirectly via libcurl in just more than a dozen packages in target. Debian Buster, the current main upstream of Apertis, includes a newer version of GnuTLS (currently 3.6.7) though upgrading to this has already been avoided due to licensing issues that will be discussed below.

- **OpenSSL**: Apertis currently provides OpenSSL version 1.1.1. This is a relatively recent release in the 1.1.1 series and is packaged as part of Debian Buster. The 1.1.1 series is currently supported[19] as an LTS release by the OpenSSL project until September 2023. Support for Debian Buster is expected[20] until June 2024.

- **NSS**: Apertis currently provides NSS version 3.42.1. This version is approximately a year and a half old, and is packaged as part of Debian Buster. As with OpenSSL, support for Debian Buster is expected until June 2024.

Some of the packages requiring TLS support only support one of the currently provided TLS implementations, often due to licensing compatibility. Other packages, most notably libraries, support multiple TLS backends, frequently including both GnuTLS and OpenSSL as options.

[16] https://www.gnutls.org/
[17] https://www.openssl.org/
[18] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS
[19] https://www.openssl.org/policies/releasestrat.html
[20] https://wiki.debian.org/LTS

Issues

The TLS libraries used in Apertis were supported, though this will not remain the case indefinitely, with Ubuntu dropping support for the currently used GnuTLS in 2022, NSS and OpenSSL 1.1 losing support in 2024.

Releases of Apertis would be expected to be based on newer versions of Debian (as covered in the Apertis Release Flow[21]. As could be expected, newer versions of Debian have integrated newer versions of these TLS libraries. Whilst upgrading to newer versions of NSS does not appear to present any issues, the GnuTLS or OpenSSL may present issues for Apertis:

GnuTLS

Whilst GnuTLS is licensed under the LGPL-2.1[22], it uses Nettle[23] and GMP[24]. Newer versions of both of these dependencies are now licensed as dual GPL-2 or LGPL-3, rather than **L**GPL-2.1.

To avoid including GnuTLS under LGPL-3 terms, should Apertis integrate a newer version it would need to be utilized under the GPL-2 terms. This would result in the binary GnuTLS library effectively being used under the terms of the GPL-2 rather than LGPL-2.1. This would restrict Apertis users from using this Apertis provided TLS implementation either directly or indirectly from any non-GPL-2 compatible applications they wish to integrate into their systems, for example in proprietary applications, where it would have the effect of requiring the app to also be GPL-2 licensed.

OpenSSL

The currently used version of OpenSSL is licensed under a custom GPL-incompatible license. OpenSSL 3.0 (the next major version of OpenSSL) will be licensed under the Apache 2.0[25] license, which is compatible with the GPL-3, but not GPL-2. This means that GPL-2 tools like `tumbler`, `connman`, `apt` or `systemd-journal-remote` cannot use the newer versions of OpenSSL without effectively becoming GPL-3 licensed or through these upstream projects applying a license exceptions (for example as OpenVPN[26] has). The OpenSSL project do not seem to hold a strong opinion on the compatibility, though suggest[27] either not using the GPL or applying an exception should you wish to gain some legal certainty.

Given the security sensitive nature of the TLS stack, utilizing unmaintained software here would be best avoided. Putting maintenance aside, these versions of

[21] https://www.apertis.org/policies/release-flow/#apertis-release-flow
[22] https://www.gnu.org/licenses/old-licenses/lgpl-2.1.en.html
[23] https://www.lysator.liu.se/~nisse/nettle/nettle.html
[24] https://gmplib.org/
[25] https://www.apache.org/licenses/LICENSE-2.0
[26] https://spdx.org/licenses/openvpn-openssl-exception.html
[27] https://www.openssl.org/docs/faq.html#LEGAL2

their respective TLS implementations may not be gaining support for any new ciphers and TLS protocol versions, which will severely limit their usefulness as time progresses. As well as not gaining newer protocol versions, the libraries may not be updated to reflect the frequently changing recommendations regarding minimal protocol versions[28] that should be supported, which may result in issues when attempting to access sites following the "Modern"recommendation. Additionally, it is likely that newer versions of the packages utilizing these TLS implementations will begin to require functionality added to newer versions of the TLS libraries thus reducing the ability of Apertis to upgrade to these too.

It is therefore imperative that a way forward is agreed upon that is acceptable to Apertis'stakeholders.

The OpenSSL project do not seem to hold a strong opinion on the compatibility, though suggest[29] either not using the GPL or applying an exception should you wish to gain some legal certainty.

The compatibility between the current OpenSSL licensing and GPL-2 is based on the premise that:

1. The OpenSSL license[30] contains licensing terms not in the GPL (such as the need to mention use of the software in all advertising material and derivatives not being able to be called OpenSSL).
2. Linking OpenSSL with a GPL-2 application creates a derivative work formed from the two pieces of code.
3. The GPL expressly states[31] that one can't "impose any further restrictions on the recipients'exercise of the rights granted herein"to the GPL licensed work.

Likewise, the Apache 2.0 license, to which version 3 of OpenSSL will be release under, contains clauses such as its patent litigation license termination clause[32].

While the argument made in step (2) is widely held by many, others disagree with this interpretation, especially when the library is dynamically linked to the application. For instance, it might be claimed[33] that a dynamically linked library is only truly combined with the application when run, not when distributed, so it would only become a derivative at that point, or it might be claimed[34] as this is the intended interface for interacting with a library this is excluded either due to fair use laws in some jurisdictions or explicitly allowed by the GPL when it states[35] "the act of running the Program is not restricted".

[28] https://wiki.mozilla.org/Security/Server_Side_TLS
[29] https://www.openssl.org/docs/faq.html#LEGAL2
[30] https://www.openssl.org/source/license-openssl-ssleay.txt
[31] https://www.gnu.org/licenses/old-licenses/gpl-2.0.html#section6
[32] http://www.apache.org/licenses/LICENSE-2.0#patent
[33] https://lwn.net/Articles/548216/
[34] https://www.linuxjournal.com/article/6366
[35] https://www.gnu.org/licenses/old-licenses/gpl-2.0.html#section0

A further argument is that the GPL states[36] "as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable". If the library is distributed as part of the OS and can be considered a major component of it, then this clause doesn't require the library to be considered as part of the software and therefore falls outside of the scope of the license. A counter argument to this is that because the application may also be considered to be distributed as part of the operating system this exception doesn't apply especially in embedded devices where the software is distributed preinstalled as a complete entity.

Most distributions seem to either ignore this potential issue or do not consider a policy to be needed. The Fedora project have deemed OpenSSL to be a system library[37] as defined by the GPL and thus there is no incompatibility. Debian historically decided that a linked library creates a derivative work and all the packages it ships should be considered a combined work, though the decision has recently been taken[38] to follow Fedora's lead here.

Alternative SSL solutions

In addition to GnuTLS and OpenSSL, there are a number of other TLS libraries available, including:

BoringSSL

BoringSSL[39] is a fork of OpenSSL being actively maintained by Google for internal use. It currently provides an OpenSSL based API, but explicitly states it comes with no API-ABI guarantees, users should expect API changes as deemed suitable for Googles internal users. BoringSSL maintains the current licensing state, though as it's developed the amount of OpenSSL-licensed code is reduced, in part through the removal of legacy code. Googles additions are currently provided under the ISC license.

LibreSSL

LibreSSL[40] is maintained by OpenBSD, it is a fork of OpenSSL v1.0.1, made as a result of the poor maintenance of OpenSSL at the time (but which has since improved). It aims to modernize the code base, improve security, and apply best practice development process. As a result of these goals a lot of legacy code has been removed. LibreSSL maintains the current licensing state,

[36] https://www.gnu.org/licenses/old-licenses/gpl-2.0.html#section3
[37] https://fedoraproject.org/wiki/Licensing:FAQ?rd=Licensing/FAQ#What.27s_the_deal_with_the_OpenSSL_license.3F
[38] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=924937#105
[39] https://boringssl.googlesource.com/boringssl/
[40] https://www.libressl.org/

with new additions provided under the ISC license. LibreSSL does not appear to have gained significant adoption which will limit the developer attention it receives.

mbed TLS

mbed TLS[41] is a TLS implementation with a small footprint, targeting embedded systems. The mbed TLS library does not provide either the OpenSSL or GnuTLS API, it provides an API at a slightly lower level, requiring more manual operations[42] and thus wrappers or porting effort would be required to use it. It is available in two versions, one distributed under the Apache-2.0 license and another separately licensed as GPL-2+, though it's understood that it will drop the GPL-2+ license in the next major release.

MesaLink

MesaLink[43] is an OpenSSL-compatible TLS library written in Rust[44]. With it being implemented in Rust it can be assumed to have some resilience due to this languages focus on safety and MesaLink recently underwent a third-party security audit with excellent results[45]. However, MesaLink only supports modern TLS standards and thus connectivity with older and less secure servers may be impacted. MesaLink is licensed as BSD-3-Clause, however it uses a large number of third party libraries, licensed as follows:

- base64: Apache-2.0/MIT
- bitflags: Apache-2.0/MIT
- env_logger: Apache-2.0/MIT
- enum_to_u8_slice_derive: BSD-3-Clause
- libc: Apache-2.0/MIT
- parking_lot: Apache-2.0/MIT
- ring: Based on BoringSSL and thus has parts licensed under the ISC and original OpenSSL licensing
- rustls: Apache-2.0/ISC/MIT
- sct: Apache-2.0/ISC/MIT
- webpki, untrusted: ISC
- webpki-roots: MPL-2.0

NSS

Network Security Services[46] (NSS) is a set of security libraries developed by Mozilla. NSS provides its own API, which is currently only supported by a

[41] https://tls.mbed.org/
[42] https://github.com/warmcat/libwebsockets/commit/9da75727858b4d60750cfcefc1673f6783e8719d
[43] https://mesalink.io/
[44] https://www.rust-lang.org/
[45] https://github.com/ctz/rustls/blob/master/audit/TLS-01-report.pdf
[46] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS

few of the applications which use TLS in Apertis, thus its use would require wrappers to be created or porting effort. It is licensed as MPL-2.0[47].

wolfSSL

The wolfSSL[48] cryptographic library provides some compatibility with OpenSSL via a compatibility header, which maps a subset of the most commonly used OpenSSL commands to its native API. It provides up-to-date standards support. wolfSSL has already been packaged in Debian. It is available under a dual license, GPL-2+ and commercial[49] licensing terms.

Possible solutions

We have considered the following options to meet Apertis'requirements.

Single stack solutions

Despite the relatively large number of TLS implementations, the required application compatibility and licensing requirements mean that there is not a single solution that will work without investing at least some development effort.

Attempting to standardize on a TLS implementation, such as by using the single stack solutions detailed below would therefore result in Apertis carrying significant changes to its packages without any guarantees that these changes could be upstreamed. These changes would thus need to be maintained as part of Apertis.

Standardize on GnuTLS, replace use of problematic dependencies
GnuTLS used to use libgcrypt as a cryptographic backend and the code is mostly structured to abstract the backend details. Reverting to using libgcrypt would result in a LGPL-2.1 licensed solution that may be viable for all desired use cases.

A preliminary investigation suggests that GnuTLS may have started to use Nettle outside of the abstracted code, which would complicate conversion back to libgcrypt. More investigation would be required to confirm this.

If libgcrypt is deemed unsuitable, an alternative may be to port GnuTLS to a different cryptographic library such as libtomcrypt (Public Domain) or libsodium (ISC). The effort required to achieve this has not been investigated.

It is likely that any resulting changes would need to be maintained as part of Apertis. It's not clear the upstream GnuTLS project would be interested in maintaining another backend.

[47] https://www.mozilla.org/en-US/MPL/2.0/
[48] https://www.wolfssl.com/
[49] https://www.wolfssl.com/license/

Standardize on an OpenSSL-compatible library As many of the applications already utilize OpenSSL, another possible approach would be writing a wrapper for a library which provides OpenSSL compatibility to also provide the GnuTLS API.

As GnuTLS itself comes with a wrapper providing OpenSSL API, it is believed that the reverse should also be possible. However, this presents some significant effort as the APIs are quite different.

An alternative approach may be to port those apps which only support GnuTLS to utilize the OpenSSL API. The effort required to achieve this has not been estimated.

Such an approach is of limited benefit as the more widely used and mature solutions providing an OpenSSL API are also licensed in such a way as to be incompatible with the GPL-2, which happens to be the license used by the most critical applications currently using GnuTLS.

Wrappering a non-GnuTLS/OpenSSL-compatible library to provide both APIs Standardizing on NSS would fall into this category. This would also be true for mbed TLS, but the Apache-2.0 license that it is future version are likely to be solely licensed under would be problematic for GPL-2-licensed applications. This option would require significant effort (creating wrappers for both GnuTLS and OpenSSL APIs) and would be a high risk strategy.

Multi-stack solutions

Attempting to choose a TLS implementation that is licensed in a manner that will work for the GPL-2-licensed applications through to Apertis'users proprietary applications massively limits the choice of library. Most of the available choices only satisfy one or other end of this spectrum, with NSS and MesaLink being the only solutions that may be suitably licensed, but which also lacks compatibility with critical applications.

As there does not appear to be any single TLS solutions meeting all use cases without significant work, we will consider keeping a multi stack solution as currently employed.

In such a scenario, a newer GnuTLS library could be allowed by accepting its dependencies under the GPL-2 license and restricting its use to places where this license wouldn't be problematic, such as existing GPL-2 software. As the existing applications written exclusively to use GnuTLS are GPL-2 or tolerant of GPL-2, this is viable.

Replace OpenSSL with compatible alternative A number of alternative TLS implementations provide an "OpenSSL-compatible"interface of one form or other. Whilst a number of these solutions are not compatible with the GPL-2, as this solution would require the continued availability of GnuTLS, the choice

of replacement can be picked without needing to provide GPL-2 compatibility. This would suggest BoringSSL, LibreSSL and MesaLink as options (wolfSSL being immediately unsuitable due to licensing).

- **BoringSSL**: Whilst actively maintained by Google for its own products, the lack of API/ABI guarantees make its adoption risky.
- **LibreSSL**: It's use inside OpenBSD suggests this will be maintained at least in the mid-term.
- **MesaLink**: As Rust is good at detecting many security related issues at compile time, its use here brings many advantages, though this needs to be weighed against its lack of support of older TLS standards which may prove problematic in some use cases.

Picking an API-compatible replacement for OpenSSL may provide a solution for the mid-term, however with OpenSSL set to release its new version at some point in the future, it is likely that we may start to see applications requiring compatibility with OpenSSL 3.0 APIs, thus requiring Apertis to reconsider its solution. Additionally, while these libraries claim OpenSSL compatibility, a different implementation may result in hard to diagnose bugs being uncovered in applications expecting OpenSSL.