taps
Internet-Draft
Intended status: Informational
Expires: 28 April 2022

M. Duke
F5 Networks, Inc.

25 October 2021

TAPS Transport Discovery
draft-duke-taps-transport-discovery-02

Abstract

The Transport Services architecture decouples applications from the protocol implementations that transport their data. While it is often straightforward to connect applications with transports that are present in the host operating system, providing a means of discovering user-installed implementations dramatically enlarges the use cases. This document discusses considerations for the design of a discovery mechanism and an example of such a design.

Discussion of this work is encouraged to happen on the TAPS IETF mailing list taps@ietf.org or on the GitHub repository which contains the draft: https://github.com/martinduke/draft-duke-taps-transport-discovery.

Discussion Venues

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

Discussion of this document takes place on the mailing list (taps@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/taps/.

Source for this draft and an issue tracker can be found at https://github.com/martinduke/draft-duke-taps-transport-discovery.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Transport Services architecture [I-D.ietf-taps-arch] enables applications to be protocol-agnostic by presenting an interface where applications can specify their required properties, and the service will select whichever protocol implementation available in the system best meets those requirements. This increases application portability and eases the introduction of new transport innovations by not requiring changes to applications.

It is sometimes straightforward for a Transport Services interface to identify the transports available in the host operating system. However, including transports installed by the user greatly expands use cases for the architecture. This document presents considerations for the secure design of a system for discovery of new protocol implementations.

Protocol Discovery would ideally have several desirable properties.

* The transport services API should not have to recompile when installing new implementations. This would not only disrupt ongoing connections, but also involve ordinary users in the complex business of downloading and building source code.

* It should support user-space implementations. Most protocol innovation begins with user space implementations, and many transports (e.g. TLS, HTTP, QUIC) are usually implemented outside the kernel long after reaching maturity.

* Protocol Discovery should not subject ordinary users to security vulnerabilities. A new protocol installation is an opportunity to hijack a user's networking stack, and Protocol Discovery requires strong protections against arbitrary code performing operations other than advertised on application data.

* Conversely, sophisticated users need a means of discovering implementations that are too new to have fully developed internet trust mechanisms. This is the only means of initially deploying new protocols for existing apps, and is the most plausible model to deploy transport services API shims for existing protocol libraries (e.g., the common TLS implementations) before their proponents deploy native support.

* Applications should not have to bring their own implementations. The Transport Services API has the concept of "framers" (see Sec. 7.1 of [I-D.ietf-taps-interface]) that provide some ability for applications to provide additional protocol encapsulation around their messages. However, one important advantage of Transport

Services is that applications do not have to rely on a third-party implementation that might not offer long term support, or add to their footprint where a functionally equivalent protocol implementation is already present on the system.

This document attempts to resolve the tension between some of these properties.

2. Conventions

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 RFC 2119 [RFC2119].

"TAPS" is an abbreviation for the transport services API.

For brevity, this document will use "app" as a shorthand for "application."

As in other TAPS documents, the concept of a "transport protocol" is expanded beyond the traditional "transport layer" to include other protocols that encapsulate application data, such as TLS, HTTP, and Websockets.

3. Entities

The Transport Services API (TAPS) is responsible for matching protocol capabilities with application requirements, and mediating further app communication with the selected protocol implementation. In this document, it actively discovers what implementations are available in the system.

The protocol implementation instantiates the transport. In this document, it offers a dynamically linked library that conforms to standard interfaces so that TAPS can interchangeability interact with it. In practice, this may be a shim layer if the underlying implementation does not support TAPS.

The protocol installer, aside from installing the implementation library and/or a TAPS shim layer, also is responsible for notifying TAPS that the implementation is present, and what its capabilities are.

Finally, the application leverages TAPS to initiate, manage, and terminate communications with other endpoints. This document does not require any changes to application behavior beyond those in the core TAPS design.

More detailed requirements for each of these entities is below.

4. Protocol Implementation

The protocol implementation must offer a dynamically linked library that offers certain APIs. TAPS SHOULD, in its documentation, provide a template for the format of these functions.

4.1. Functions

The objects below need not follow the semantics of the TAPS application API. In particular, a "message" is unlikely to have all the property information described there, instead being a more primitive buffer in which raw data is stored.

''' Listener := Listen(localEndpoint) '''

Listen opens a socket and listens on the specified address, and returns a handle to the resulting listener.

''' Listener.Stop() '''

Stop causes the listener to stop accepting connections. Subsequent events will return handles to the resulting connection.

''' Connection.Send(Message) Connection.Receive(Message) '''

TAPS will provide a Message object for the protocol to either send, or use to store incoming data.

Further APIs are TBD.

4.2. Events

The protocol needs to throw all the events described in the TAPS Application API, although the return values may not exactly conform to the same semantics.

TAPS SHOULD provide an event framework that frees the protocol implementation from running its own thread for a polling loop. TAPS also SHOULD account for the possibility that the implementation may have its own polling architecture. If true, the protocol MUST conform to the API by translating its events into the signals or callbacks that TAPS expects.

5. Protocol Installer

The installer might use the operating system's package manager or "app store", or be a simple script. Besides installing the implementation, the installer also writes data to a registry that TAPS will access to discover the implementation.

This data will include:

* the name of the supported protocol(s);

* optionally, the versions of those protocols;

* the path to the implementations TAPS-compliant library;

* the properties that the protocol implementation supports, as described in Section 4.2 of [I-D.ietf-taps-interface]; and

* information to authenticate the entry (see Section 7).

Of course, a de-installer should remove the appropriate registry entry.

A TAPS implementation SHOULD provide a template for this registry information.

One potential instantiation of this would have protocol installers write a file to a directory that, in a specified markup language, described the information above.

6. TAPS

TAPS creates a registry for protocol implementations, which might be a database or a directory. To prevent inadvertent security vulnerabilities, the host system SHOULD, at minimum, require administrative privileges to write to the registry.

No later than upon receipt of request for a Preconnection, TAPS MUST access the registry to determine the available protocols and their properties. It is perfectly valid for there to be multiple implementations of a protocol.

TAPS SHOULD validate entries in the registry using the provided authentication data.

One potential instaniation would start daemon that monitored the status of the registry. Upon any change to the registry, the daemon might:

* authenticate any new entry in accordance with security policy;

* verify that the required function handles are present;

* run tests to verify the installation's claimed properties;

* inform the user of the new protocol, requesting permission to trust it; and

* write the information into shared memory for the use of Preconnections.

7. Security Considerations

User-space installation of protocols provides enormous opportunities for attackers to hijack a network stack. While this has always been possible with arbitrary protocol implementations, with TAPS applications completely unaware of the installation can be victims of such an attack.

An implementation might advertise properties it does not actually provide to attract more traffic. For example, a "TLS" implementation might not encrypt anything at all. A TAPS implementation MAY run tests on newly installed protocols to verify it provides the advertised properties.

Moreover, in principle an implementation could deliver application data anywhere it wanted with little visibility to the application, much less the user.

The origin of the protocol installer is important to the trust model. Obviously, transports in the kernel do not introduce vulnerabilities specific to TAPS. A trusted package manager (e.g. the Apple App Store or yum) may imply a minimal level of veracity of the available packages. Protocol implementations directly downloaded from the internet without mediation through these mechanisms require the greatest care.

Ongoing work on this document will largely focus on building mechanisms to mitigate this weakness. Some promising approaches include:

* administrative privileges to alter the TAPS registry;

* a special certificate authority that provides an authentication of the implementation's explicit and implicit claims, as well as the integrity of the installed binary;

* each installer generates a private key and provides the corresponding public key, so that only possessors of the private key can modify or delete the registry entry;

* confirmation by a human, prominently warned of potential consequences, if the installation is not mediated through a trusted authority.

8. IANA Considerations

This document has no IANA requirements.

9. Implementation Status

> *RFC Editor's Note:* Please remove this section prior to publication of a final version of this document.

The Dynamic TAPS project (https://github.com/f5networks/dynamic-taps) is a preliminary effort to implement the concepts in this document.

10. Informative References

[I-D.ietf-taps-arch]
Pauly, T., Trammell, B., Brunstrom, A., Fairhurst, G., Perkins, C., Tiesel, P. S., and C. A. Wood, "An Architecture for Transport Services", Work in Progress, Internet-Draft, draft-ietf-taps-arch-11, 12 July 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-taps-arch-11>.

[I-D.ietf-taps-interface]
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P. S., Wood, C. A., Pauly, T., and K. Rose, "An Abstract Application Layer Interface to Transport Services", Work in Progress, Internet-Draft, draft-ietf-taps-interface-13, 12 July 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-taps-interface-13>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://doi.org/10.17487/RFC2119>.

Appendix A. Acknowledgments

Tim Worsley contributed important ideas to this document.

Appendix B. Change Log

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

B.1. since draft-duke-taps-transport-discovery-01

* Added output of initial implementation work

Author's Address

Martin Duke
F5 Networks, Inc.

Email: martin.h.duke@gmail.com