Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 16, 2018

M. Thomson
Mozilla

September 12, 2017

Long-term Viability of Protocol Extension Mechanisms
draft-thomson-use-it-or-lose-it-00

Abstract

The ability to change protocols depends on exercising the extension and version negotiation mechanisms that support change. Protocols that don't use these mechanisms can find that deploying changes can be difficult and costly.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 16, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

A successful protocol [SUCCESS] will change in ways that allow it to continue to fulfill the needs of its users. New use cases, conditions and constraints on the deployment of a protocol can render a protocol that does not change obsolete.

Usage patterns and requirements for a protocol shift over time. Protocols can react to these shifts in one of three ways: adjust usage patterns within the constraints of the protocol, extend the protocol, and replace the protocol. These reactions are progressively more disruptive, but are also dictated by the nature of the change in requirements over longer periods.

Experience with Internet-scale protocol deployment shows that changing protocols is not uniformly successful. [TRANSITIONS] examines the problem more broadly.

This document examines the specific conditions that determine whether protocol maintainers have the ability to design and deploy new or modified protocols. Section 4 outlines several strategies that might aid in ensuring that protocol changes remain possible over time.

2. Implementations of Protocols are Imperfect

A change to a protocol can be made extremely difficult to deploy if there are bugs in implementations with which the new deployment needs to interoperate. Bugs in the handling of new codepoints or

extensions can mean that instead of handling the mechanism as designed, endpoints react poorly. This can manifest as abrupt termination of sessions, errors, crashes, or disappearances of endpoints and timeouts.

Interoperability with other implementations is usually highly valued, so deploying mechanisms that trigger adverse reactions like these can be untenable. Where interoperability is a competitive advantage, this is true even if the negative reactions happen infrequently or only under relatively rare conditions.

Deploying a change to a protocol could require fixing a substantial proportion of the bugs that the change exposes. This can involve a difficult process that includes identifying the cause of these errors, finding the responsible implementation, coordinating a bug fix and release plan, contacting the operator of affected services, and waiting for the fix to be deployed to those services.

Given the effort involved in fixing these problems, the existence of these sorts of bugs can outright prevent the deployment of some types of protocol changes. It could even be necessary to come up with a new protocol design that uses a different method to achieve the same result.

2.1. Good Protocol Design is Not Sufficient

It is often argued that the design of a protocol extension point or version negotiation capability is critical to the freedom that it ultimately offers.

RFC 6709 [EXTENSIBILITY] contains a great deal of well-considered advice on designing for extension. It includes the following advice:

> This means that, to be useful, a protocol version- negotiation mechanism should be simple enough that it can reasonably be assumed that all the implementers of the first protocol version at least managed to implement the version-negotiation mechanism correctly.

This has proven to be insufficient in practice. Many protocols have evidence of imperfect implementation of these critical mechanisms. Mechanisms that aren't used are the ones that fail most often. The same paragraph from RFC 6709 acknowledges the existence of this problem, but does not offer any remedy:

> The nature of protocol version-negotiation mechanisms is that, by definition, they don't get widespread real-world testing until

after the base protocol has been deployed for a while, and its deficiencies have become evident.

Transport Layer Security (TLS) [TLS12] provides examples of where a design that is objectively sound fails when incorrectly implemented. TLS provides examples of failures in protocol version negotiation and extensibility.

Version negotiation in TLS 1.2 and earlier uses the "Highest mutually supported version (HMSV)" scheme exactly as it is described in [EXTENSIBILITY]. However, clients are unable to advertise a new version without causing a non-trivial proportions of sessions to fail due to bugs in server and middlebox implementations.

Intolerance to new TLS versions is so severe [INTOLERANCE] that TLS 1.3 [TLS13] has abandoned HMSV version negotiation for a new mechanism.

The server name indication (SNI) [TLS-EXT] in TLS is another excellent example of the failure of a well-designed extensibility point. SNI uses the same technique for extension that is used with considerable success in other parts of the TLS protocol. The original design of SNI includes the ability to include multiple names of different types.

What is telling in this case is that SNI was defined with just one type of name: a domain name. No other type has ever been standardized, though several have been proposed. Despite an otherwise exemplary design, SNI is so inconsistently implemented that any hope for using the extension point it defines has been abandoned [SNI].

2.2. Multi-Party Interactions and Middleboxes

Even the most superficially simple protocols can often involve more actors than is immediately apparent. A two-party protocol still has two ends, and even at the endpoints of an interaction, protocol elements can be passed on to other entities in ways that can affect protocol operation.

One of the key challenges in deploying new features in a protocol is ensuring compatibility with all actors that could influence the outcome.

Protocols that deploy without active measures against intermediation can accrue middleboxes that depend on certain aspects of the protocol [PATH-SIGNALS]. In particular, one of the consequences of an unencrypted protocol is that any element on path can interact with

the protocol. For example, HTTP was specifically designed with intermediation in mind, transparent proxies [HTTP] are not only possible but sometimes advantageous, despite some significant downsides. Consequently, transparent proxies for cleartext HTTP are commonplace.

Middleboxes are also protocol participants, to the degree that they are able to observe and act in ways that affect the protocol. The degree to which a middlebox participates varies from the basic functions that a router performs to full participation. For example, a SIP back-to-back user agent (B2BUA) [B2BUA] can be very deeply involved in the SIP protocol.

By increasing the number of different actors involved in any single protocol exchange, the number of potential implementation bugs that a deployment needs to contend with also increases. In particular, incompatible changes to a protocol that might be negotiated between endpoints in ignorance of the presence of a middlebox can result in a middlebox acting badly.

Thus, middleboxes can increase the difficulty of deploying changes to a protocol considerably.

3. Retaining Viable Protocol Evolution Mechanisms

If design is insufficient, what then would give protocol designers the freedom to later change a deployed protocol?

Michel Foucault defines freedom as a practice rather than a state that is bestowed or attained:

> Freedom is practice; [...] the freedom of men is never assured by the laws and the institutions that are intended to guarantee them. [...] I think it can never be inherent in the structure of things to guarantee the exercise of freedom. The guarantee of freedom is freedom. -[FOUCAULT]

In the same way, the design of a protocol for extensibility and eventual replacement [EXTENSIBILITY] does not guarantee the ability to exercise those options.

3.1. Practice Can Ensure Viability

Planning and careful specificiation of mechanisms that support protocol evolution is a necessary precondition for their later availability. However, whether those mechanisms are available for use depends on their correct implementation and deployment. The

nature of a protocol deployment has a significant effect on whether that protocol can be changed.

The fact that the freedom to change depends on practice is evident in protocols that are known to have viable version negotiation or extension points. The definition of mechanisms alone is insufficient; it's the active use of those mechanisms that determines the existence of freedom.

For example, header fields in email [SMTP], HTTP [HTTP] and SIP [SIP] all derive from the same basic design. There is no evidence of significant barriers to deploying header fields with new names and semantics in email and HTTP, though the widespread deployment of SIP B2BUAs means that new SIP header fields can be more difficult.

In another example, the attribute-value pairs (AVPs) in Diameter [DIAMETER] are fundamental to the design of the protocol. The definition of new uses of Diameter regularly exercise the ability to add new AVPs and do so with no fear that the new feature might not be successfully deployed.

These examples show extension points that are heavily used also being relatively unaffected by deployment issues preventing addition of new values for new use cases.

These examples also confirm the case that good design is not a prerequisite for success. On the contrary, success is often despite shortcomings in the design. For instance, the shortcomings of HTTP header fields are significant enough that there are ongoing efforts to improve the syntax [HTTP-HEADERS].

Only using a protocol capability is able to ensure availability of that capability. Protocols that fail to use a mechanism, or a protocol that only rarely uses a mechanism, suffer an inability to rely on that mechanism.

3.2. Dependency is Better

The best way to guarantee that a protocol mechanism is used is to make it critical to an endpoint participating in that protocol. This means that implementations rely on both the existence of the protocol mechanism and its use.

For example, the message format in SMTP relies on header fields for most of its functions, including the most basic functions. A deployment of SMTP cannot avoid including an implementation of header field handling. In addition to this, the regularity with which new header fields are defined and used ensures that deployments

frequently encounter header fields that it does not understand. An SMTP implementation therefore needs to be able to both process header fields that it understands and ignore those that it does not.

In this way, implementing the extensibility mechanism is not merely mandated by the specification, it is critical to the functioning of a protocol deployment. Should an implementation fail to correctly implement the mechanism, that failure would quickly become apparent.

Caution is advised to avoid assuming that this is sufficient to ensure extensibility in the long term. If the set of possible variations is small and deployments do not change over time, implementations might not see new variations. Those implementations might still exhibit errors when presented with a new variation.

3.3. Unused Extension Points Become Unusable

In contrast, there are many examples of extension points in protocols that have been either completely unused, or their use was so infrequent that they could no longer be relied upon to function correctly.

HTTP has a number of very effective extension points in addition to the aforementioned header fields. It also has some examples of extension point that are so rarely used that it is possible that they are not at all usable. Extension points in HTTP that might be unwise to use include the extension point on each chunk in the chunked transfer coding [HTTP], the ability to use transfer codings other than the chunked coding, and the range unit in a range request [HTTP-RANGE].

4. Defensive Design Principles for Protocols

There are several potential approaches that can provide some measure of protection against a protocol deployment becoming resistant to change.

4.1. Active Use

As discussed in Section 3, the most effective defense against misuse of protocol extension points is active use.

4.2. Grease

"Grease" [GREASE] identifies lack of use as an issue (protocol mechanisms "rusting" shut) and proposes a system of use that exercises extension points by using dummy values.

The primary feature of the grease design is aimed at the style of negotiation most used in TLS, where the client offers a set of options and the server chooses the one that it most prefers from those that it supports. A client that uses grease randomly offers options (usually just one) from a set of reserved values. These values are guaranteed to never be assigned real meaning, so the server will never have cause to genuinely select one of these values.

The principle that grease operates on is that an implementation that is regularly exposed to unknown values is not likely to become intolerant of new values when they appear. This depends somewhat on the fact that the difficulty of implementing the protocol mechanism correctly is not significantly more effort than implementing code to specifically filter out the randomized grease values.

To avoid simple techniques for filtering greasing codepoints, grease values are not reserved from a single contiguous block of code points, but are distributed evenly across the entire space of code points. Reserving a randomly selected set of code points has a greater chance of avoiding this problem, though it might be more difficult to specify and implement, especially over larger code point spaces.

Without reserved greasing codepoints, an implementation can use code points from spaces used for private or experimental use if such a range exists. In addition to the risk of triggering participation in an unwanted experiment, this can be less effective. Incorrect implementations might still be able to correctly identify these code points and ignore them.

Grease is deployed with the intent of quickly detecting errors in implementing the mechanisms it safeguards. Any failure to properly handle grease values is more likely to be detected.

This form of defensive design has some limitations. It does not necessarily create the need for an implementation to rely on the mechanism it safeguards; that is determined by the underlying protocol itself. More critically, it does not easily translate to other forms of extension point. Other techniques might be necessary for protocols that don't rely on the particular style of exchange that is predominant in TLS.

For instance, grease works poorly for HMSV negotiation, where offering a higher version risks acceptance of a newly deployed version.

4.3. Cryptography

Cryptography can be used to reduce the number of entities that can participate in a protocol. Using tools like TLS ensures that only authorized participants are able to influence whether a new protocol feature is used.

Data that is exchanged under encryption cannot be seen by middleboxes, excluding them from participating in that part of the protocol. Similarly, data that is exchanged with integrity protection cannot be modified by middleboxes.

The QUIC protocol [QUIC] adopts both encryption and integrity protection. Encryption is used to carefully control what information is exposed to middleboxes. QUIC also uses integrity protection over all the data it exchanges to prevent modification.

4.4. Visibility of Faults

Modern software engineering practice includes a strong emphasis on measuring the effects of changes and correcting based on that feedback. Runtime monitoring of system health is an important part of that, which relies on systems of logging and synthetic health indicators, such as aggregate transaction failure rates.

Feedback is critical to the success of the grease technique (see Section 4.2). The system only works if an implementer creates a way to ensure that errors are detected and analyzed. This process can be automated, but when operating at scale it might be difficult or impossible to collect details of specific errors.

Treating errors in protocol implementation as fatal can greatly improve visibility. Disabling automatic recovery from protocol errors can be disruptive to users when those errors occur, but it also ensures that errors are made visible.

Visibility of error conditions is especially important if users are part of the feedback system.

New protocol designs are encouraged to define conditions that result in fatal errors. Competitive pressures often force implementations to favor strategies that mask or hide errors. Standardizing on error handling that ensures visibility of flaws avoids handling that suppresses problems.

Feedback on errors is more important during the development and early deployment of a change. Disabling automatic error recovery methods during development improves visibility of errors.

Automated feedback systems are important for automated systems, or where error recovery is also automated. For instance, connection failures with HTTP alternative services [ALT-SVC] are not permitted to affect the outcome of transactions. A feedback system for capturing failures in alternative services is therefore crucial to ensuring that failures are detected and the mechanism remains viable.

5. Security Considerations

The ability to design, implement, and deploy new protocol mechanisms can be critical to security. In particular, it is important to be able to replace cryptographic algorithms over time [AGILITY].

6. IANA Considerations

This document makes no request of IANA.

7. Informative References

[AGILITY] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

[ALT-SVC] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/info/rfc7838>.

[B2BUA] Kaplan, H. and V. Pascual, "A Taxonomy of Session Initiation Protocol (SIP) Back-to-Back User Agents", RFC 7092, DOI 10.17487/RFC7092, December 2013, <https://www.rfc-editor.org/info/rfc7092>.

[DIAMETER] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn, Ed., "Diameter Base Protocol", RFC 6733, DOI 10.17487/RFC6733, October 2012, <https://www.rfc-editor.org/info/rfc6733>.

[EXTENSIBILITY] Carpenter, B., Aboba, B., Ed., and S. Cheshire, "Design Considerations for Protocol Extensions", RFC 6709, DOI 10.17487/RFC6709, September 2012, <https://www.rfc-editor.org/info/rfc6709>.

[FOUCAULT] Foucault, M. and P. Rabinow, Ed., "The Foucault Reader", ISBN 0394713400, November 1984.

[GREASE] Benjamin, D., "Applying GREASE to TLS Extensibility", draft-ietf-tls-grease-00 (work in progress), January 2017.

[HTTP] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[HTTP-HEADERS] Kamp, P., "HTTP Header Common Structure", draft-ietf-httpbis-header-structure-01 (work in progress), April 2017.

[HTTP-RANGE] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <https://www.rfc-editor.org/info/rfc7233>.

[INTOLERANCE] Kario, H., "Re: [TLS] Thoughts on Version Intolerance", July 2016, <https://mailarchive.ietf.org/arch/msg/tls/bOJ2JQc3HjAHFFWCiNTIb0JuMZc>.

[PATH-SIGNALS] Hardie, T., "Path signals", draft-hardie-path-signals-01 (work in progress), May 2017.

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-05 (work in progress), August 2017.

[SIP] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[SMTP] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[SNI] Langley, A., "Accepting that other SNI name types will never work", March 2016, <https://mailarchive.ietf.org/arch/msg/tls/1t79gzNItZd71DwwoaqcQQ_4Yxc>.

[SUCCESS] Thaler, D. and B. Aboba, "What Makes for a Successful Protocol?", RFC 5218, DOI 10.17487/RFC5218, July 2008, <https://www.rfc-editor.org/info/rfc5218>.

[TLS-EXT] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[TLS12] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-21 (work in progress), July 2017.

[TRANSITIONS] Thaler, D., Ed., "Planning for Protocol Adoption and Subsequent Transitions", RFC 8170, DOI 10.17487/RFC8170, May 2017, <https://www.rfc-editor.org/info/rfc8170>.

Author's Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com