Internet Architecture Board B. Carpenter
Internet-Draft B. Aboba (ed)
Intended Status: Informational S. Cheshire
Expires: December 10, 2012 9 June 2012

Design Considerations for Protocol Extensions
draft-iab-extension-recs-14

Abstract

This document discusses architectural issues related to the extensibility of Internet protocols, with a focus on design considerations. It is intended to assist designers of both base protocols and extensions. Case studies are included. A companion document, RFC 4775/BCP 125, discusses procedures issues relating to the extensibility of IETF protocols.

Copyright Notice

Table of Contents

1. Introduction

When developing protocols, IETF Working Groups (WGs) often include mechanisms whereby these protocols can be extended in the future. It is often a good principle to design extensibility into protocols; as described in "What Makes for a Successful Protocol" [RFC5218], a "wildly successful" protocol is one that becomes widely used in ways not originally anticipated. Well-designed extensibility mechanisms facilitate the evolution of protocols and help make it easier to roll out incremental changes in an interoperable fashion. However, at the same time experience has shown that extensions carry the risk of unintended consequences, such as interoperability issues, operational problems or security vulnerabilities.

The proliferation of extensions, even well designed ones, can be costly. As noted in "Simple Mail Transfer Protocol" [RFC5321] Section 2.2.1:

> Experience with many protocols has shown that protocols with few options tend towards ubiquity, whereas protocols with many options tend towards obscurity.
>
> Each and every extension, regardless of its benefits, must be carefully scrutinized with respect to its implementation, deployment, and interoperability costs.

This is hardly a recent concern. "TCP Extensions Considered Harmful" [RFC1263] was published in 1991. "Extend" or "extension" occurs in the title of more than 400 existing Request For Comment (RFC) documents. Yet generic extension considerations have not been documented previously.

The purpose of this document is to describe the architectural principles of sound extensibility design, in order to minimize such risks. Formal procedures for extending IETF protocols are discussed in "Procedures for Protocol Extensions and Variations" BCP 125 [RFC4775].

The rest of this document is organized as follows: Section 2 discusses routine and major extensions. Section 3 describes architectural principles for protocol extensibility. Section 4 explains how designers of base protocols can take steps to anticipate and facilitate the creation of such subsequent extensions in a safe and reliable manner.

Readers are advised to study the whole document, since the considerations are closely linked.

1.1. Requirements Language

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 BCP 14, RFC 2119 [RFC2119].

2. Routine and Major Extensions

The risk of unintended consequences from an extension is especially high if the extension is performed by a different team than the original designers, who may stray outside implicit design constraints or assumptions. As a result, it is highly desirable for the original designers to articulate the design constraints and assumptions, so as to enable extensions to be done carefully and with a full understanding of the base protocol, existing implementations, and current operational practice.

To assist extension designers and reviewers, protocol documents should provide guidelines explaining how extensions should be performed, and guidance on the appropriate use of protocol extension mechanisms should be developed.

Protocol components that are designed with the specific intention of allowing extensibility should be clearly identified, with specific and complete instructions on how to extend them. This includes the process for adequate review of extension proposals: do they need community review and if so how much and by whom?

The level of review required for protocol extensions will typically vary based on the nature of the extension. Routine extensions may require minimal review, while major extensions may require wide review. Guidance on which extensions may be considered 'routine' and which ones are 'major' are provided in the sections that follow.

2.1. What Constitutes a Major Extension?

Major extensions may have characteristics leading to a risk of interoperability failures, security vulnerabilities or operational problems. Where these characteristics are present, it is necessary to pay close attention to backward compatibility with implementations and deployments of the unextended protocol, and to the potential for inadvertent introduction of security or operational exposures.

Extension designers should examine their design for the following issues:

1. Modifications or extensions to the underlying protocol. An

extension document should be considered to update the underlying protocol specification if an implementation of the underlying protocol would need to be updated to accommodate the extension. This should not be necessary if the underlying protocol was designed with a modular interface. Examples of extensions modifying the underlying protocol include specification of additional transports (see Section 4.6), changing protocol semantics or defining new message types that may require implementation changes in existing and deployed implementations of the protocol, even if they do not want to make use of the new functions. A base protocol that does not uniformly permit "silent discard" of unknown extensions may automatically enter this category, even for apparently minor extensions. Handling of "unknown" extensions is discussed in more detail in Section 4.7.

2. Changes to the basic architectural assumptions. This may include architectural assumptions that are explicitly stated or those that have been assumed by implementers. For example, this would include adding a requirement for session state to a previously stateless protocol.

3. New usage scenarios not originally intended or investigated. This can potentially lead to operational difficulties when deployed, even in cases where the "on-the-wire" format has not changed. For example, the level of traffic carried by the protocol may increase substantially, packet sizes may increase, and implementation algorithms that are widely deployed may not scale sufficiently or otherwise be up to the new task at hand. For example, a new DNS Resource Record (RR) type that is too big to fit into a single UDP packet could cause interoperability problems with existing DNS clients and servers. Similarly, the additional traffic that results from an extension to a routing protocol could have a detrimental impact on the performance or stability of implementations that do not implement the extension.

4. Changes to the extension model. Adverse impacts are very likely if the base protocol contains an extension mechanism and the proposed extension does not fit into the model used to create and define that mechanism. Extensions that have the same properties as those that were anticipated when an extension mechanism was devised are much less likely to be disruptive than extensions that don't fit the model. Also, changes to the extension model itself (including changes limiting further extensibility) can create interoperability problems.

5. Changes to protocol syntax. Changes to protocol syntax bring with them the potential for backward compatibility issues. If at all possible, extensions should be designed for compatibility with

existing syntax, so as to avoid interoperability failures.

6. Interrelated extensions to multiple protocols. A set of interrelated extensions to multiple protocols typically carries a greater danger of interoperability issues or incompatibilities than a simple extension. Consequently, it is important that such proposals receive earlier and more in-depth review than unitary extensions.

7. Changes to the security model. Changes to the protocol security model (or even addition of new security mechanisms within an existing framework) can introduce security vulnerabilities or adversely impact operations. Consequently, it is important that such proposals undergo security as well as operational review. Security considerations are discussed in Section 5.

8. Performance impact. An extension that impacts performance can have adverse consequences, particularly if the performance of existing deployments is affected.

2.2. When is an Extension Routine?

An extension may be considered 'routine' if it does not meet the criteria for being considered a 'major' extension and if its handling is opaque to the protocol itself (e.g. does not substantially change the pattern of messages and responses). For this to apply, no changes to the base protocol can be required, nor can changes be required to existing and currently deployed implementations, unless they make use of the extension. Furthermore, existing implementations should not be impacted. This typically requires that implementations be able to ignore 'routine' extensions without ill-effects.

Examples of routine extensions include the Dynamic Host Configuration Protocol (DHCP) vendor-specific option [RFC2132], Remote Authentication Dial In User Service (RADIUS) Vendor-Specific Attributes [RFC2865], the enterprise Object IDentifier (OID) tree for Management Information Base (MIB) modules and vendor Multipurpose Internet Mail Extension (MIME) types. Such extensions can safely be made with minimal discussion.

Processes that allow routine extensions with minimal or no review (such as the "First Come First Served" (FCFS) allocation policy described in "Guidelines for Writing an IANA Considerations Section in RFCs" [RFC5226]) should be used sparingly. In particular, they should be limited to cases that are unlikely to result in interoperability problems, or security or operational exposures.

Experience has shown that even routine extensions may benefit from review by experts. For example, even though DHCP carries opaque data, defining a new option using completely unstructured data may lead to an option that is unnecessarily hard for clients and servers to process.

3. Architectural Principles

This section describes basic principles of protocol extensibility:

1. Extensibility features should be limited to what is reasonably anticipated when the protocol is developed.

2. Protocol extensions should be designed for global interoperability.

3. Protocol extensions should be architecturally compatible with the base protocol.

4. Protocol extension mechanisms should not be used to create incompatible protocol variations.

5. Extension mechanisms need to be testable.

6. Protocol parameter assignments need to be coordinated to avoid potential conflicts.

7. Extensions to critical components require special care. A critical component is one whose failure can lead to Internet-wide reliability and security issues or performance degradation.

3.1. Limited Extensibility

Protocols should not be made more extensible than clearly necessary at inception, in order to enable optimization along dimensions (e.g., bandwidth, state, memory requirements, deployment time, latency, etc.) important to the most common use cases.

The process for defining new extensibility mechanisms should ensure that adequate review of proposed extensions will take place before widespread adoption.

As noted in "What Makes for a Successful Protocol" [RFC5218], "wildly successful" protocols far exceed their original goals, in terms of scale, purpose (being used in scenarios far beyond the initial design), or both. This implies that all potential uses may not be known at inception. As a result, extensibility mechanisms may need to be revisited as additional use cases reveal themselves. However,

this does not imply that an initial design needs to take all potential needs into account at inception.

3.2. Design for Global Interoperability

As noted in [RFC4775] Section 3.1:

> According to its Mission Statement [RFC3935], the IETF produces high quality, relevant technical and engineering documents, including protocol standards. The mission statement goes on to say that the benefit of these standards to the Internet "is in interoperability - that multiple products implementing a standard are able to work together in order to deliver valuable functions to the Internet's users".
>
> One consequence of this mission is that the IETF designs protocols for the single Internet. The IETF expects its protocols to work the same everywhere. Protocol extensions designed for limited environments may be reasonable provided that products with these extensions interoperate with products without the extensions. Extensions that break interoperability are unacceptable when products with and without the extension are mixed. It is the IETF's experience that this tends to happen on the Internet even when the original designers of the extension did not expect this to happen.
>
> Another consequence of this definition of interoperability is that the IETF values the ability to exchange one product implementing a protocol with another. The IETF often specifies mandatory-to-implement functionality as part of its protocols so that there is a core set of functionality sufficient for interoperability that all products implement. The IETF tries to avoid situations where protocols need to be profiled to specify which optional features are required for a given environment, because doing so harms interoperability on the Internet as a whole.

Since the global Internet is more than a collection of incompatible protocols (or "profiles") for use in separate private networks, implementers supporting extensions in shipping products or multi-site experimental usage must assume that systems will need to interoperate on the global Internet.

A key requirement for interoperable extension design is that the base protocol must be well designed for interoperability, and that extensions must have unambiguous semantics. Ideally, the protocol mechanisms for extension and versioning should be sufficiently well described that compatibility can be assessed on paper. Otherwise, when two "private" or "experimental" extensions encounter each other

on a public network, unexpected interoperability problems may occur. However, as noted in the TLS case study (see Appendix A.3), it is not sufficient to design extensibility carefully; it also must be implemented carefully.

3.2.1. Private Extensions

Experience shows that separate private networks often end up using equipment from the same vendors, or end up having portable equipment like laptop computers move between them, and networks that were originally envisaged as being separate can end up being connected later.

Consider a "private" extension installed on a work computer which, being portable, is sometimes connected to a home network or a hotel network. If the "private" extension is incompatible with an unextended version of the same protocol, problems will occur.

Similarly, problems can occur if "private" extensions conflict with each other. For example, imagine the situation where one site chose to use DHCP [RFC2132] option code 62 for one meaning, and a different site chose to use DHCP option code 62 for a completely different, incompatible, meaning. It may be impossible for a vendor of portable computing devices to make a device that works correctly in both environments.

One approach to solving this problem has been to reserve parts of an identifier namespace for "limited applicability" or "site-specific" use, such as "X-" headers in email messages [RFC822] or "P-" headers in SIP [RFC3427]. However, as noted in "Deprecating the X- Prefix and Similar Constructs in Application Protocols" Appendix B [XDASH]:

> The primary problem with the "X-" convention is that unstandardized parameters have a tendency to leak into the protected space of standardized parameters, thus introducing the need for migration from the "X-" name to a standardized name. Migration, in turn, introduces interoperability issues (and sometimes security issues) because older implementations will support only the "X-" name and newer implementations might support only the standardized name. To preserve interoperability, newer implementations simply support the "X-" name forever, which means that the unstandardized name has become a de facto standard (thus obviating the need for segregation of the name space into standardized and unstandardized areas in the first place).

As a result, the notion of "X-" headers was removed from the Internet Message Format standard when it was updated in 2001 [RFC2822] and within SIP, [RFC5727] Section 4 deprecated the guidance provided in

[RFC3427] on the creation of "P-" headers. More generally, as noted in [XDASH] Section 1:

> This document generalizes from the experience of the email and SIP communities by doing the following:
>
> 1. Deprecates the "X-" convention for newly-defined parameters in application protocols, even where that convention was only implicit instead of being codified in a protocol specification (as was done for email in [RFC822]).

3.2.2. Local Use

Values designated as "experimental" or "local use" are only appropriate for use in a limited set of circumstances such as for use in early implementations of an extension restricted to a single site.

For example, "Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP and TCP Headers" [RFC4727] discusses experimental values for IP and transport headers, and "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers" [RFC2474] defines experimental/local use ranges for differentiated services code points.

Such values should be used with care and only for their stated purpose: experiments and local use. They are unsuitable for Internet-wide use, since they may be used for conflicting purposes and thereby cause interoperability failures. Packets containing experimental or local use values must not be allowed out of the domain in which they are meaningful.

"Assigning Experimental and Testing Numbers Considered Useful" BCP 82 [RFC3692] Section 1 provides guidance on the use of experimental code points:

> Numbers in the experimentation range ... are not intended to be used in general deployments or be enabled by default in products or other general releases. In those cases where a product or release makes use of an experimental number, the end user must be required to explicitly enable the experimental feature and likewise have the ability to chose and assign which number from the experimental range will be used for a specific purpose (i.e., so the end user can ensure that use of a particular number doesn't conflict with other on-going uses). Shipping a product with a specific value pre-enabled would be inappropriate and can lead to interoperability problems when the chosen value collides with a different usage, as it someday surely will.

> From the above, it follows that it would be inappropriate for a group of vendors, a consortia, or another Standards Development Organization to agree among themselves to use a particular value for a specific purpose and then agree to deploy devices using those values. By definition, experimental numbers are not guaranteed to be unique in any environment other than one where the local system administrator has chosen to use a particular number for a particular purpose and can ensure that a particular value is not already in use for some other purpose.
>
> Once an extension has been tested and shown to be useful, a permanent number could be obtained through the normal assignment procedures.

However, as noted in [XDASH] Appendix B, assigning a parameter block for experimental use is only necessary when the parameter pool is limited:

> "Assigning Experimental and Testing Numbers Considered Useful" ... implies that the "X-" prefix is also useful for experimental parameters. However, BCP 82 addresses the need for protocol numbers when the pool of such numbers is strictly limited (e.g., DHCP options) or when a number is absolutely required even for purely experimental purposes (e.g., the Protocol field of the IP header). In almost all application protocols that make use of protocol parameters (including email headers, media types, HTTP headers, vCard parameters and properties, URNs, and LDAP field names), the name space is not limited or constrained in any way, so there is no need to assign a block of names for private use or experimental purposes ...
>
> Therefore it appears that segregating the parameter space into a standardized area and a unstandardized area has few if any benefits, and has at least one significant cost in terms of interoperability.

3.2.3. Multi-site Experiments

Where an experiment is undertaken among a diverse set of experimental sites connected via the global Internet, the use of "experimental" or "local use" code points is inadvisable. This might include, for example, sites that take a prototype implementation of some protocol and use that both within their site but, importantly, among the full set of other sites interested in that protocol. In such a situation it is impractical and probably impossible to coordinate the de-confliction of "experimental" code points. As noted in [RFC5226] Section 4.1:

> For private or local use ... No attempt is made to prevent multiple sites from using the same value in different (and incompatible) ways ... assignments are not generally useful for broad interoperability. It is the responsibility of the sites making use of the Private Use range to ensure that no conflicts occur (within the intended scope of use).

HIP and LISP are examples where a set of experimental sites are collaborating among themselves, but not necessarily in a tightly coordinated way. Both HIP and LISP have dealt with this by having unique non-experimental code points allocated to HIP and LISP, respectively, at time of publication of their respective Experimental RFCs.

3.3. Architectural Compatibility

Since protocol extension mechanisms may impact interoperability, it is important that they be architecturally compatible with the base protocol.

This includes understanding what current implementations do and how a proposed extension will interact with deployed systems. Is it clear when a proposed extension (or its proposed usage) will operationally stress existing implementations or the underlying protocol itself if widely deployed? If this is not explained in the base protocol specification, is this covered in an extension design guidelines document?

As part of the definition of new extension mechanisms, it is important to address whether the mechanisms make use of features as envisaged by the original protocol designers, or whether a new extension mechanism is being invented. If a new extension mechanism is being invented, then architectural compatibility issues need to be addressed.

To assist in the assessment of architectural compatibility, protocol documents should provide guidelines explaining how extensions should be performed, and guidance on the appropriate use of protocol extension mechanisms should be developed.

Protocol components that are designed with the specific intention of allowing extensibility should be clearly identified, with specific and complete instructions on how to extend them. This includes the process for adequate review of extension proposals: do they need community review and if so how much and by whom?

Documents relying on extension mechanisms need to explicitly identify the mechanisms being relied upon. For example, a document defining

new data elements should not implicitly define new data types or protocol operations without explicitly describing those dependencies and discussing their impact. Where extension guidelines are available, mechanisms need to indicate whether they are compliant with those guidelines and if not, why not.

Examples of extension guidelines documents include:

1. "Guidelines for Extending the Extensible Provisioning Protocol (EPP)" [RFC3735], which provides guidelines for use of EPP's extension mechanisms to define new features and object management capabilities.

2. "Guidelines for Authors and Reviewers of MIB Documents" BCP 111 [RFC4181], which provides guidance to protocol designers creating new MIB modules.

3. "Guidelines for Authors of Extensions to the Session Initiation Protocol (SIP)" [RFC4485], which outlines guidelines for authors of SIP extensions.

4. "Considerations for Lightweight Directory Access Protocol (LDAP) Extensions" BCP 118 [RFC4521], which discusses considerations for designers of LDAP extensions.

5. "RADIUS Design Guidelines" BCP 158 [RFC6158], which provides guidelines for the design of attributes used by the Remote Authentication Dial In User Service (RADIUS) protocol.

3.4. Protocol Variations

Protocol variations - specifications that look very similar to the original but don't interoperate with each other or with the original - are even more harmful to interoperability than extensions. In general, such variations should be avoided. Causes of protocol variations include incompatible protocol extensions, uncoordinated protocol development, and poorly designed "profiles".

Designing a protocol for extensibility may have the perverse side effect of making it easy to construct incompatible extensions. Protocol extension mechanisms should not be used to create incompatible forks in development. An extension may lead to interoperability failures unless the extended protocol correctly supports all mandatory and optional features of the unextended base protocol, and implementations of the base protocol operate correctly in the presence of the extensions. In addition, it is necessary for an extension to interoperate with other extensions.

As noted in "Uncoordinated Protocol Development Considered Harmful" [RFC5704] Section 1, incompatible forks in development can result from the uncoordinated adaptation of a protocol, parameter or code-point:

> In particular, the IAB considers it an essential principle of the protocol development process that only one SDO maintains design authority for a given protocol, with that SDO having ultimate authority over the allocation of protocol parameter code-points and over defining the intended semantics, interpretation, and actions associated with those code-points.

Note that problems can occur even when one SDO maintains design authority, if protocol parameter code-points are reused. As an example, both RFC 5421 [RFC5421] and RFC 5422 [RFC5422] reused previously assigned EAP type codes. As described in the IESG note in [RFC5421]:

> The reuse of previously assigned EAP Type Codes is incompatible with EAP method negotiation as defined in RFC 3748.

3.4.1. Profiles

Profiling is a common technique for improving interoperability within a target environment or set of scenarios. Generally speaking, there are two approaches to profiling:

a) Removal or downgrading of normative requirements (thereby creating potential interoperability problems);

b) Elevation of normative requirement levels (such as from a MAY/SHOULD to a MUST). This can be done in order to improve interoperability by narrowing potential implementation choices (such as when the underlying protocol is ill-defined enough to permit non-interoperable yet compliant implementations), or to meet specific operational requirements (such as enabling use of stronger cryptographic mechanisms than those mandated in the specification).

While approach a) is potentially harmful, approach b) may be beneficial.

In order to avoid creating interoperability problems when profiled implementations interact with others over the Global Internet, profilers need to remain cognizant of the implications of removing normative requirements. As noted in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119] Section 6, imperatives are to be used with care, and as a result, their removal within a profile is likely to result in serious consequences:

> Imperatives of the type defined in this memo must be used with care and sparingly. In particular, they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions) For example, they must not be used to try to impose a particular method on implementors where the method is not required for interoperability.

As noted in [RFC2119] Sections 3 and 4, recommendations cannot be removed from profiles without serious consideration:

> there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

Even the removal of optional features and requirements can have consequences. As noted in [RFC2119] Section 5, implementations which do not support optional features still retain the obligation to ensure interoperation with implementations that do:

> An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

3.5. Testability

Experience has shown that it is insufficient merely to correctly specify extensibility and backwards compatibility in an RFC. It is also important that implementations respect the compatibility mechanisms; if not, non-interoperable pairs of implementations may arise. The TLS case study (Appendix A.3) shows how important this can be.

In order to determine whether protocol extension mechanisms have been properly implemented, testing is required. However, for this to be possible, test cases need to be developed. If a base protocol document specifies extension mechanisms but does not utilize them or provide examples, it may not be possible to develop effective test cases based on the base protocol specification alone. As a result, base protocol implementations may not be properly tested and non-compliant extension behavior may not be detected until these implementations are widely deployed.

To encourage correct implementation of extension mechanisms, base protocol specifications should clearly articulate the expected behavior of extension mechanisms and should include examples of correct extension behavior.

3.6. Protocol Parameter Registration

As noted in [RFC4775] Section 3.2:

> An extension is often likely to make use of additional values added to an existing IANA registry ... It is essential that such new values are properly registered by the applicable procedures, including expert review where applicable ... Extensions may even need to create new IANA registries in some cases.
>
> Experience shows that the importance of this is often underestimated during extension design; designers sometimes assume that a new codepoint is theirs for the asking, or even simply for the taking.

Before creating a new protocol parameter registry, existing registries should be examined to determine one of them can be used instead (see http://www.iana.org/protocols/).

To avoid conflicting usage of the same registry value, as well as to prevent potential difficulties in determining and transferring parameter ownership, it is essential that all new values are registered. If this is not done, there is nothing to prevent two different extensions picking the same value. When these two extensions "meet" each other on the Internet, failure is inevitable.

A surprisingly common case of this is misappropriation of assigned Transmission Control Protocol (TCP) (or User Datagram Protocol (UDP)) registered port numbers. This can lead to a client for one service attempting to communicate with a server for another service. Another common case is the use of unregistered URI schemes. Numerous cases could be cited, but not without embarrassing specific implementers. For general rules see [RFC5226], and for specific rules and registries see the individual protocol specification RFCs and the IANA web site.

While in theory a "standards track" or "IETF consensus" parameter allocation policy may be instituted to encourage protocol parameter registration or to improve interoperability, in practice problems can arise if the procedures result in so much delay that requesters give up and "self-allocate" by picking presumably-unused code points. Where self-allocation is prevalent, the information contained within registries may become inaccurate, particularly when third parties are

prohibited from updating entries so as to improve accuracy. In these situations, it is important to consider whether registration processes need to be changed to support the role of a registry as "documentation of how the Internet is operating".

3.7. Extensions to Critical Protocols

Some protocols (such as Domain Name Service (DNS), Border Gateway Protocol (BGP)) or algorithms (such as congestion control) have become critical components of the Internet infrastructure. A critical component is one whose failure can lead to Internet-wide reliability and security issues or performance degradation. When such protocols or algorithms are extended, the potential exists for negatively impacting the reliability and security of the global Internet.

As a result, special care needs to be taken with these extensions, such as taking explicit steps to isolate existing uses from new ones. For example, this can be accomplished by requiring the extension to utilize a different port or multicast address, or by implementing the extension within a separate process, without access to the data and control structures of the base protocol.

Experience has shown that even when a mechanism has proven benign in other uses, unforeseen issues may result when adding it to a critical protocol. For example, both ISIS and OSPF support opaque Link State Attributes (LSAs) which are propagated by intermediate nodes that don't understand the LSA. Within Interior Gateway Protocols (IGPs), support for opaque LSAs has proven useful without introducing instability.

However, within BGP, 'attribute tunneling' has resulted in large scale routing instabilities, since remote nodes may reset the LOCAL session if the tunneled attributes are malformed or aren't understood. This has required modification to BGP error handling, as noted in "Error Handling for Optional Transitive Attribute BGP Attributes" [Transitive].

In general, when extending protocols with local failure conditions, tunneling of attributes that may trigger failures in non-adjacent nodes should be avoided. This is particularly problematic when the originating node receives no indicators of remote failures it may have triggered.

4. Considerations for the Base Protocol

Good extension design depends on a well-designed base protocol. To promote interoperability, designers should:

1. Ensure a well-written base protocol specification. Does the base protocol specification make clear what an implementer needs to support and does it define the impact that individual operations (e.g., a message sent to a peer) will have when invoked?

2. Design for backward compatibility. Does the base protocol specification describe how to determine the capabilities of a peer, and negotiate the use of extensions? Does it indicate how implementations handle extensions that they do not understand? Is it possible for an extended implementation to negotiate with an unextended peer to find a common subset of useful functions?

3. Respect underlying architectural or security assumptions. Is there a document describing the underlying architectural assumptions, as well as considerations that have arisen in operational experience? Or are there undocumented considerations that have arisen as the result of operational experience, after the original protocol was published?

For example, will backward compatibility issues arise if extensions reverse the flow of data, allow formerly static parameters to be changed on the fly, or change assumptions relating to the frequency of reads/writes?

4. Minimize impact on critical infrastructure. For a protocol that represents a critical element of Internet infrastructure, it is important to explain when it is appropriate to isolate new uses of the protocol from existing ones.

For example, is it explained when a proposed extension (or usage) has the potential for negatively impacting critical infrastructure to the point where explicit steps would be appropriate to isolate existing uses from new ones?

5. Provide guidance on data model extensions. Is there a document that explains when a protocol extension is routine and when it represents a major change?

For example, is it clear when a data model extension represents a major versus a routine change? Are there guidelines describing when an extension (such as a new data type) is likely to require a code change within existing implementations?

4.1. Version Numbers

Any mechanism for extension by versioning must include provisions to ensure interoperability, or at least clean failure modes. Imagine

someone creating a protocol and using a "version" field and populating it with a value (1, let's say), but giving no information about what would happen when a new version number appears in it. This would be a bad protocol design and description; it should be clear what the expectation is and how it can be tested. For example, stating that 1.X must be compatible with any version 1 code, but version 2 or greater is not expected to be compatible, has different implications than stating that version 1 must be a proper subset of version 2.

An example of an under-specified versioning mechanism is provided by the MIME-Version header, originally defined in "MIME (Multipurpose Internet Mail Extensions)" [RFC1341]. As noted in [RFC1341] Section 1:

> A MIME-Version header field ... uses a version number to declare a message to be conformant with this specification and allows mail processing agents to distinguish between such messages and those generated by older or non-conformant software, which is presumed to lack such a field.

Beyond this, [RFC1341] provided little guidance on versioning behavior, or even the format of the MIME-Version header, which was specified to contain "text". [RFC1521] which obsoleted [RFC1341], better defined the format of the version field, but still did not clarify the versioning behavior:

> Thus, future format specifiers, which might replace or extend "1.0", are constrained to be two integer fields, separated by a period. If a message is received with a MIME-version value other than "1.0", it cannot be assumed to conform with this specification ...
>
> It is not possible to fully specify how a mail reader that conforms with MIME as defined in this document should treat a message that might arrive in the future with some value of MIME-Version other than "1.0". However, conformant software is encouraged to check the version number and at least warn the user if an unrecognized MIME- version is encountered.

Thus, even though [RFC1521] defined a MIME-Version header with a syntax suggestive of a "Major/Minor" versioning scheme, in practice the MIME-Version header was little more than a decoration.

A better example is ROHC (Robust Header Compression). ROHCv1 [RFC3095] supports a certain set of profiles for compression algorithms. But experience had shown that these profiles had limitations, so the ROHC WG developed ROHCv2 [RFC5225]. A ROHCv1

implementation does not contain code for the ROHCv2 profiles. As the ROHC WG charter said during the development of ROHCv2:

> It should be noted that the v2 profiles will thus not be compatible with the original (ROHCv1) profiles, which means less complex ROHC implementations can be realized by not providing support for ROHCv1 (over links not yet supporting ROHC, or by shifting out support for ROHCv1 in the long run). Profile support is agreed through the ROHC channel negotiation, which is part of the ROHC framework and thus not changed by ROHCv2.

Thus in this case both backwards-compatible and backwards-incompatible deployments are possible. The important point is a clearly thought out approach to the question of operational compatibility. In the past, protocols have utilized a variety of strategies for versioning, many of which have proven problematic. These include:

1. No versioning support. This approach is exemplified by Extensible Authentication Protocol (EAP) [RFC3748] as well as Remote Authentication Dial In User Service (RADIUS) [RFC2865], both of which provide no support for versioning. While lack of versioning support protects against the proliferation of incompatible dialects, the need for extensibility is likely to assert itself in other ways, so that ignoring versioning entirely may not be the most forward thinking approach.

2. Highest mutually supported version (HMSV). In this approach, implementations exchange the version numbers of the highest version each supports, with the negotiation agreeing on the highest mutually supported protocol version. This approach implicitly assumes that later versions provide improved functionality, and that advertisement of a particular version number implies support for all lower version numbers. Where these assumptions are invalid, this approach breaks down, potentially resulting in interoperability problems. An example of this issue occurs in Protected Extensible Authentication Protocol [PEAP] where implementations of higher versions may not necessarily provide support for lower versions.

3. Assumed backward compatibility. In this approach, implementations may send packets with higher version numbers to legacy implementations supporting lower versions, but with the assumption that the legacy implementations will interpret packets with higher version numbers using the semantics and syntax defined for lower versions. This is the approach taken by Port-Based Access Control [IEEE-802.1X]. For this approach to work, legacy implementations need to be able to accept packets of known types

with higher protocol versions without discarding them; protocol enhancements need to permit silent discard of unsupported extensions; implementations supporting higher versions need to refrain from mandating new features when encountering legacy implementations.

4. Major/minor versioning. In this approach, implementations with the same major version but a different minor version are assumed to be backward compatible, but implementations are required to negotiate a mutually supported major version number. This approach assumes that implementations with a lower minor version number but the same major version can safely ignore unsupported protocol messages.

5. Min/max versioning. This approach is similar to HMSV, but without the implied obligation for clients and servers to support all versions back to version 1, in perpetuity. It allows clients and servers to cleanly drop support for early versions when those versions become so old that they are no longer relevant and no longer required. In this approach, the client initiating the connection reports the highest and lowest protocol versions it understands. The server reports back the chosen protocol version:

a. If the server understands one or more versions in the client's range, it reports back the highest mutually understood version.

b. If there is no mutual version, then the server reports back some version that it does understand (selected as described below). The connection is then typically dropped by client or server, but reporting this version number first helps facilitate useful error messages at the client end:

* If there is no mutual version, and the server speaks any version higher than client max, it reports the lowest version it speaks which is greater than the client max. The client can then report to the user, "You need to upgrade to at least version <xx>."

* Else, the server reports the highest version it speaks. The client can then report to the user, "You need to request the server operator to upgrade to at least version <min>."

Protocols generally do not need any version-negotiation mechanism more complicated than the mechanisms described here. 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. 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.

4.2. Reserved Fields

Protocols commonly include one or more "reserved" fields, clearly intended for future extensions. It is good practice to specify the value to be inserted in such a field by the sender (typically zero) and the action to be taken by the receiver when seeing some other value (typically no action). In packet format diagrams, such fields are typically labeled "MBZ", to be read as, "Must Be Zero on transmission, Must Be Ignored on reception."

A common mistake of inexperienced protocol implementers is to think that "MBZ" means that it's their software's job to verify that the value of the field is zero on reception, and reject the packet if not. This is a mistake, and such software will fail when it encounters future versions of the protocol where these previously reserved fields are given new defined meanings. Similarly, protocols should carefully specify how receivers should react to unknown extensions (headers, TLVs etc.), such that failures occur only when that is truly the intended outcome.

4.3. Encoding Formats

Using widely-supported encoding formats leads to better interoperability and easier extensibility.

As described in "IAB Thoughts on Encodings for International Domain Names" [RFC6055], the number of encodings should be minimized and complex encodings are generally a bad idea. As soon as one moves outside the ASCII repertoire, issues relating to collation, string valid code points, encoding, normalization and comparison arise that extensions must handle with care. See [draft-iab-identifier-comparison], [draft-ietf-precis-problem-statement] and [draft-ietf-precis-framework].

An example is the Simple Network Management Protocol (SNMP) Structure of Managed Information (SMI). Guidelines exist for defining the Management Information Base (MIB) objects that SNMP carries [RFC4181]. Also, multiple textual conventions have been published, so that MIB designers do not have to reinvent the wheel when they need a commonly encountered construct. For example, the "Textual Conventions for Internet Network Addresses" [RFC4001] can be used by any MIB designer needing to define objects containing IP addresses, thus ensuring consistency as the body of MIBs is extended.

4.4. Parameter Space Design

In some protocols the parameter space is either infinite (e.g., Header field names) or sufficiently large that it is unlikely to be exhausted. In other protocols, the parameter space is limited, and in some cases, has proven inadequate to accommodate demand. Common mistakes include:

a. A version field that is too small (e.g., two bits or less). When designing a version field, existing as well as potential versions of a protocol need to be taken into account. For example, if a protocol is being standardized for which there are existing implementations with known interoperability issues, more than one version for "pre-standard" implementations may be required. If two "pre-standard" versions are required in addition to a version for an IETF standard, then a two-bit version field would only leave one additional version code-point for a future update, which could be insufficient. This problem was encountered during the development of the PEAPv2 protocol [PEAP].

b. A small parameter space (e.g., 8-bits or less) along with a First Come, First Served (FCFS) allocation policy. In general, an FCFS allocation policy is only appropriate in situations where parameter exhaustion is highly unlikely. In situations where substantial demand is anticipated within a parameter space, the space should either be designed to be sufficient to handle that demand, or vendor extensibility should be provided to enable vendors to self-allocate. The combination of a small parameter space, an FCFS allocation policy, and no support for vendor extensibility is particularly likely to prove ill-advised. An example of such a combination was the design of the original 8-bit EAP Method Type space [RFC2284].

Once the potential for parameter exhaustion becomes apparent, it is important that it be addressed as quickly as possible. Protocol changes can take years to appear in implementations and by then the exhaustion problem could become acute.

Options for addressing a protocol parameter exhaustion problem include:

Rethinking the allocation regime
: Where it becomes apparent that the size of a parameter space is insufficient to meet demand, it may be necessary to rethink the allocation mechanism, in order to prevent or delay parameter space exhaustion. In revising parameter allocation mechanisms, it is important to consider both supply and demand aspects so as to avoid unintended consequences such as self-allocation or the development of black markets for the re-sale of protocol parameters.

For example, a few years after approval of RFC 2284 [RFC2284], it became clear that the combination of a FCFS allocation policy and lack of support for vendor-extensions had created the potential for exhaustion of the EAP Method Type space within a few years. To address the issue, [RFC3748] Section 6.2 changed the allocation policy for EAP Method Types from FCFS to Expert Review, with Specification Required. Since this allocation policy revision did not change the demand for EAP Method Types, it would have been likely to result in self-allocation within the standards space, had mechanisms not been provided to expand the method type space (including support for vendor-specific method types).

Support for vendor-specific parameters

If the demand that cannot be accommodated is being generated by vendors, merely making allocation harder could make things worse if this encourages vendors to self-allocate, creating interoperability problems. In such a situation, support for vendor-specific parameters should be considered, allowing each vendor to self-allocate within their own vendor-specific space based on a vendor's Private Enterprise Code (PEC). For example, in the case of the EAP Method Type space, [RFC3748] Section 6.2 also provided for an Expanded Type space for "functions specific only to one vendor's implementation".

Extensions to the parameter space

If the goal is to stave off exhaustion in the face of high demand, a larger parameter space may be helpful; this may require a new version of the protocol (such as was required for IPv6). Where vendor-specific parameter support is available, this may be achieved by allocating a PEC for IETF use. Otherwise it may be necessary to try to extend the size of the parameter fields, which could require a new protocol version or other substantial protocol changes.

Parameter reclamation

In order to gain time, it may be necessary to reclaim unused parameters. However, it may not be easy to determine whether a parameter that has been allocated is in use or not, particularly if the entity that obtained the allocation no longer exists or has been acquired (possibly multiple times).

Parameter Transfer

When all the above mechanisms have proved infeasible and parameter exhaustion looms in the near future, enabling the transfer of ownership of protocol parameters can be considered as a means for improving allocation efficiency. However, enabling transfer of parameter ownership can be far from simple if the parameter allocation process was not originally designed to enable title

searches and ownership transfers.

A parameter allocation process designed to uniquely allocate code-points is fundamentally different from one designed to enable title search and transfer. If the only goal is to ensure that a parameter is not allocated more than once, the parameter registry will only need to record the initial allocation. On the other hand, if the goal is to enable transfer of ownership of a protocol parameter, then it is important not only to record the initial allocation, but also to track subsequent ownership changes, so as to make it possible to determine and transfer title. Given the difficulty of converting from a unique allocation regime to one requiring support for title search and ownership transfer, it is best for the desired capabilities to be carefully thought through at the time of registry establishment.

4.5. Cryptographic Agility

Extensibility with respect to cryptographic algorithms is desirable in order to provide resilience against the compromise of any particular algorithm. "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management" BCP 132 [RFC4962] Section 3 provides some basic advice:

> The ability to negotiate the use of a particular cryptographic algorithm provides resilience against compromise of a particular cryptographic algorithm ... This is usually accomplished by including an algorithm identifier and parameters in the protocol, and by specifying the algorithm requirements in the protocol specification. While highly desirable, the ability to negotiate key derivation functions (KDFs) is not required. For interoperability, at least one suite of mandatory-to-implement algorithms MUST be selected ...
>
> This requirement does not mean that a protocol must support both public-key and symmetric-key cryptographic algorithms. It means that the protocol needs to be structured in such a way that multiple public-key algorithms can be used whenever a public-key algorithm is employed. Likewise, it means that the protocol needs to be structured in such a way that multiple symmetric-key algorithms can be used whenever a symmetric-key algorithm is employed.

In practice, the most difficult challenge in providing cryptographic agility is providing for a smooth transition in the event that a mandatory-to-implement algorithm is compromised. Since it may take significant time to provide for widespread implementation of a previously undeployed alternative, it is often advisable to recommend

implementation of alternative algorithms of distinct lineage in addition to those made mandatory-to-implement, so that an alternative algorithm is readily available. If such a recommended alternative is not in place, then it would be wise to issue such a recommendation as soon as indications of a potential weakness surface. This is particularly important in the case of potential weakness in algorithms used to authenticate and integrity-protect the cryptographic negotiation itself, such as KDFs or message integrity checks (MICs). Without secure alternatives to compromised KDF or MIC algorithms, it may not be possible to secure the cryptographic negotiation while retaining backward compatibility.

4.6. Transport

In the past, IETF protocols have been specified to operate over multiple transports. Often the protocol was originally specified to utilize a single transport, but limitations were discovered in subsequent deployment, so that additional transports were subsequently specified.

In a number of cases, the protocol was originally specified to operate over UDP, but subsequent operation disclosed one or more of the following issues, leading to the specification of alternative transports:

a. Payload fragmentation (often due to the introduction of extensions or additional usage scenarios);

b. Problems with congestion control, transport reliability or efficiency;

c. Lack of deployment in multicast scenarios, which had been a motivator for UDP transport.

On the other hand, there are also protocols that were originally specified to operate over reliable transport that have subsequently defined transport over UDP, due to one or more of the following issues:

d. NAT traversal concerns that were more easily addressed with UDP transport;

e. Scalability problems, which could be improved by UDP transport.

Since specification of a single transport offers the highest potential for interoperability, protocol designers should carefully consider not only initial but potential future requirements in the selection of a transport protocol. Where UDP transport is selected,

the guidance provided in "Unicast UDP Usage Guidelines for Application Designers" [RFC5405] should be taken into account.

After significant deployment has occurred, there are few satisfactory options for addressing problems with the originally selected transport protocol. While specification of additional transport protocols is possible, removal of a widely implemented transport protocol is likely to result in interoperability problems and should be avoided.

Mandating support for the initially selected transport protocol, while designating additional transport protocols as optional may have limitations. Since optional transport protocols are typically introduced due to the advantages they afford in certain scenarios, in those situations implementations not supporting optional transport protocols may exhibit degraded performance or may even fail.

While mandating support for multiple transport protocols may appear attractive, designers need to realistically evaluate the likelihood that implementers will conform to the requirements. For example, where resources are limited (such as in embedded systems), implementers may choose to only support a subset of the mandated transport protocols, resulting in non-interoperable protocol variants.

4.7. Handling of Unknown Extensions

IETF protocols have utilized several techniques for handling of unknown extensions. One technique (often used for vendor-specific extensions) is to specify that unknown extensions be "silently discarded".

While this approach can deliver a high level of interoperability, there are situations in which it is problematic. For example, where security functionality is involved, "silent discard" may not be satisfactory, particularly if the recipient does not provide feedback as to whether it supports the extension or not. This can lead to operational security issues that are difficult to detect and correct, as noted in Appendix A.2 and "common RADIUS Implementation Issues and Suggested Fixes" [RFC5080] Section 2.5.

In order to ensure that a recipient supports an extension, a recipient encountering an unknown extension may be required to explicitly reject it and to return an error, rather than proceeding. This can be accomplished via a "Mandatory" bit in a TLV-based protocol such as L2TP [RFC2661], or a "Require" or "Proxy-Require" header in a text-based protocol such as SIP [RFC3261] or HTTP [RFC2616].

Since a mandatory extension can result in an interoperability failure when communicating with a party that does not support the extension, this designation may not be permitted for vendor-specific extensions, and may only be allowed for standards-track extensions. To enable fallback operation with degraded functionality, it is good practice for the recipient to indicate the reason for the failure, including a list of unsupported extensions. The initiator can then retry without the offending extensions.

Typically only the recipient will find itself in the position of rejecting a mandatory extension, since the initiator can explicitly indicate which extensions are supported, with the recipient choosing from among the supported extensions. This can be accomplished via an exchange of TLVs, such as in IKEv2 [RFC5996] or Diameter [RFC3588], or via use of "Accept", "Accept-Encoding", "Accept-Language", "Allow" and "Supported" headers in a text-based protocol such as SIP [RFC3261] or HTTP [RFC2616].

5. Security Considerations

An extension must not introduce new security risks without also providing adequate counter-measures, and in particular it must not inadvertently defeat security measures in the unextended protocol. Thus, the security analysis for an extension needs to be as thorough as for the original protocol - effectively it needs to be a regression analysis to check that the extension doesn't inadvertently invalidate the original security model.

This analysis may be simple (e.g., adding an extra opaque data element is unlikely to create a new risk) or quite complex (e.g., adding a handshake to a previously stateless protocol may create a completely new opportunity for an attacker).

When the extensibility of a design includes allowing for new and presumably more powerful cryptographic algorithms to be added, particular care is needed to ensure that the result is in fact increased security. For example, it may be undesirable from a security viewpoint to allow negotiation down to an older, less secure algorithm.

6. IANA Considerations

[RFC Editor: please remove this section prior to publication.]

This document has no IANA Actions.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4775] Bradner, S., Carpenter, B., and T. Narten, "Procedures for Protocol Extensions and Variations", BCP 125, RFC 4775, December 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

7.2. Informative References

[XDASH] Saint-Andre, P., Crocker, D. and M. Nottingham, "Deprecating the X- Prefix and Similar Constructs in Application Protocols", draft-ietf-appsawg-xdash-05.txt, Internet draft (work in progress), April 2012.

[IEEE-802.1X] Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Port-Based Network Access Control", IEEE Standard 802.1X-2004, December 2004.

[PEAP] Palekar, A., Simon, D., Salowey, J., Zhou, H., Zorn, G. and S. Josefsson, "Protected EAP Protocol (PEAP) Version 2", draft-josefsson-pppext-eap-tls-eap-10.txt, Expired Internet draft (work in progress), October 2004.

[RFC822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[RFC1263] O'Malley, S. and L. Peterson, "TCP Extensions Considered Harmful", RFC 1263, October 1991.

[RFC1341] Freed, N. and N. Borenstein, "MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1341, June 1992.

[RFC1521] Borenstein, N. and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, September 1993.

[RFC2058] Rigney, C., Rubens, A., Simpson, W. and S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC 2058,

January 1997.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, March 1997.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2284] Blunk, L. and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)", RFC 2284, March 1998.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, August 1999.

[RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)",RFC 2671, August 1999.

[RFC2822] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC2882] Mitton, D., "Network Access Servers Requirements: Extended RADIUS Practices", RFC 2882, July 2000.

[RFC3095] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T., and H. Zheng, "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095, July 2001.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3427] Mankin, A., Bradner, S., Mahy, R., Willis, D., Ott, J., and B. Rosen, "Change Process for the Session Initiation Protocol

(SIP)", BCP 67, RFC 3427, December 2002.

[RFC3575] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)", RFC 3575, July 2003.

[RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G. and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.

[RFC3597] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, September 2003.

[RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, January 2004.

[RFC3735] Hollenbeck, S., "Guidelines for Extending the Extensible Provisioning Protocol (EPP)", RFC 3735, March 2004.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J. and H. Lefkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC3935] Alvestrand, H., "A Mission Statement for the IETF", RFC 3935, October 2004.

[RFC4001] Daniele, M., Haberman, B., Routhier, S., and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 4001, February 2005.

[RFC4181] Heard, C., "Guidelines for Authors and Reviewers of MIB Documents", BCP 111, RFC 4181, September 2005.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[RFC4485] Rosenberg, J. and H. Schulzrinne, "Guidelines for Authors of Extensions to the Session Initiation Protocol (SIP)", RFC 4485, May 2006.

[RFC4521] Zeilenga, K., "Considerations for Lightweight Directory Access Protocol (LDAP) Extensions", BCP 118, RFC 4521, June 2006.

[RFC4727] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, November 2006.

[RFC4929] Andersson, L. and A. Farrel, "Change Process for Multiprotocol Label Switching (MPLS) and Generalized MPLS (GMPLS) Protocols and Procedures", BCP 129, RFC 4929, June 2007.

[RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.

[RFC5080] Nelson, D. and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes", RFC 5080, December 2007.

[RFC5218] Thaler, D., and B. Aboba, "What Makes for a Successful Protocol?", RFC 5218, July 2008.

[RFC5225] Pelletier, G. and K. Sandlund, "RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite", RFC 5225, April 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", RFC 5405 (BCP 145), November 2008.

[RFC5421] Cam-Winget, N. and H. Zhou, "Basic Password Exchange within the Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5421, March 2009.

[RFC5422] Cam-Winget, N., McGrew, D., Salowey, J. and H. Zhou, "Dynamic Provisioning Using Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5422, March 2009.

[RFC5704] Bryant, S. and M. Morrow, "Uncoordinated Protocol Development Considered Harmful", RFC 5704, November 2009.

[RFC5727] Peterson, J., Jennings, C. and R. Sparks, "Change Process for the Session Initiation Protocol (SIP) and the Real-time Applications and Infrastructure Area", BCP 67, RFC 5727, March 2010.

[RFC5996] Kaufman, C., Hoffman, P., Nir, Y. and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.

[RFC6055] Thaler, D., Klensin, J. and S. Cheshire, "IAB Thoughts on Encodings for Internationalized Domain Names", RFC 6055, February 2011.

[RFC6158] DeKok, A. and G. Weber, "RADIUS Design Guidelines", BCP 158, RFC 6158, March 2011.

[Transitive]
Scudder, J., Chen, E., Mohapatra, P. and K. Patel, "Revised Error Handling for BGP UPDATE Messages", Internet draft (work in progress), draft-ietf-idr-optional-transitive-04, October, 2011.

Acknowledgments

This document is heavily based on an earlier draft under a different title by Scott Bradner and Thomas Narten.

That draft stated: The initial version of this document was put together by the IESG in 2002. Since then, it has been reworked in response to feedback from John Loughney, Henrik Levkowetz, Mark Townsley, Randy Bush and others.

Valuable comments and suggestions on the current form of the document were made by Loa Andersson, Leslie Daigle, Phillip Hallam-Baker, Ted Hardie, Alfred Hoenes, John Klensin, Eric Rescorla, Adam Roach, Pekka Savola, Alan DeKok, Roy Fielding, Barry Leiba, Stewart Bryant and Ran Atkinson. The text on TLS experience was contributed by Yngve Pettersen.

IAB Members at the Time of Approval

Bernard Aboba
Jari Arkko
Marc Blanchet
Ross Callon
Alissa Cooper
Spencer Dawkins
Joel Halpern
Russ Housley
David Kessens
Danny McPherson
Jon Peterson
Dave Thaler
Hannes Tschofenig

Appendix A. Examples

This section discusses some specific examples, as case studies.

A.1. Already documented cases

There are certain documents that specify a change process or describe extension considerations for specific IETF protocols:

The SIP change process [RFC3427], [RFC4485], [RFC5727]
The (G)MPLS change process (mainly procedural) [RFC4929]
LDAP extensions [RFC4521]
EPP extensions [RFC3735]
DNS extensions [RFC2671][RFC3597]
SMTP extensions [RFC5321]

It is relatively common for MIBs, which are all in effect extensions of the SMI data model, to be defined or extended outside the IETF. BCP 111 [RFC4181] offers detailed guidance for authors and reviewers.

A.2. RADIUS Extensions

The RADIUS [RFC2865] protocol was designed to be extensible via addition of Attributes to a Data Dictionary on the server, without requiring code changes. However, this extensibility model assumed that Attributes would conform to a limited set of data types and that vendor extensions would be limited to use by vendors, in situations in which interoperability was not required. Subsequent developments have stretched those assumptions.

From the beginning, uses of the RADIUS protocol extended beyond the scope of the original protocol definition (and beyond the scope of the RADIUS Working Group charter). In addition to rampant self-allocation within the limited RADIUS standard attribute space, vendors defined their own RADIUS commands. This lead to the rapid proliferation of vendor-specific protocol variants. To this day, many common implementation practices have not been documented. As noted in "Extended RADIUS Practices" [RFC2882] Section 1:

> The RADIUS Working Group was formed in 1995 to document the protocol of the same name, and was chartered to stay within a set of bounds for dial-in terminal servers. Unfortunately the real world of Network Access Servers (NASes) hasn't stayed that small and simple, and continues to evolve at an amazing rate.
>
> This document shows some of the current implementations on the market have already outstripped the capabilities of the RADIUS protocol. A quite a few features have been developed completely

> outside the protocol. These features use the RADIUS protocol structure and format, but employ operations and semantics well beyond the RFC documents.

The limited set of data types defined in [RFC2865] has lead to subsequent documents defining new data types. Since new data types are typically defined implicitly as part of defining a new attribute, and because RADIUS client and server implementations differ in their support of these additional specifications, there is no definitive registry of RADIUS data types and data type support has been inconsistent. To catalog commonly implemented data types as well as to provide guidance for implementers as well as attribute designers, "RADIUS Design Guidelines" [RFC6158] Section 2.1 includes advice on basic and complex data types. Unfortunately, these guidelines were published 14 years after the RADIUS protocol was first documented in [RFC2058].

Section 6.2 of the RADIUS specification [RFC2865] defines a mechanism for Vendor-Specific extensions (Attribute 26), and states that use of Vendor-Specific extensions:

> should be encouraged instead of allocation of global attribute types, for functions specific only to one vendor's implementation of RADIUS, where no interoperability is deemed useful.

However, in practice usage of Vendor-Specific Attributes (VSAs) has been considerably broader than this. In particular, VSAs have been used by Standards Development Organizations (SDOs) to define their own extensions to the RADIUS protocol. This has caused a number of problems.

One issue concerns the data model for VSAs. Since it was not envisaged that multi-vendor VSA implementations would need to interoperate, the RADIUS specification [RFC2865] does not define the data model for VSAs, and allows multiple sub-attributes to be included within a single Attribute of type 26. Since this enables VSAs to be defined which would not be supportable by current implementations if placed within the standard RADIUS attribute space, this has caused problems in standardizing widely deployed VSAs, as discussed in "RADIUS Design Guidelines" BCP 158 [RFC6158] Section 2.4:

> RADIUS attributes can often be developed within the vendor space without loss (and possibly even with gain) in functionality. As a result, translation of RADIUS attributes developed within the vendor space into the standard space may provide only modest benefits, while accelerating the exhaustion of the standard space. We do not expect that all RADIUS attribute specifications

> requiring interoperability will be developed within the IETF, and allocated from the standard space. A more scalable approach is to recognize the flexibility of the vendor space, while working toward improvements in the quality and availability of RADIUS attribute specifications, regardless of where they are developed.
>
> It is therefore NOT RECOMMENDED that specifications intended solely for use by a vendor or SDO be translated into the standard space.

Another issue is how implementations should handle unknown VSAs. [RFC2865] Section 5.26 states:

> Servers not equipped to interpret the vendor-specific information sent by a client MUST ignore it (although it may be reported). Clients which do not receive desired vendor-specific information SHOULD make an attempt to operate without it, although they may do so (and report they are doing so) in a degraded mode.

However, since VSAs do not contain a "mandatory" bit, RADIUS clients and servers may not know whether it is safe to ignore unknown VSAs. For example, in the case where VSAs pertain to security (e.g., Filters), it may not be safe to ignore them. As a result, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes" [RFC5080] Section 2.5 includes the following caution:

> To avoid misinterpretation of service requests encoded within VSAs, RADIUS servers SHOULD NOT send VSAs containing service requests to RADIUS clients that are not known to understand them. For example, a RADIUS server should not send a VSA encoding a filter without knowledge that the RADIUS client supports the VSA.

In addition to extending RADIUS by use of VSAs, SDOs have also defined new values of the Service-Type attribute in order to create new RADIUS commands. Since the RADIUS specification [RFC2865] defined Service-Type values as being allocated First Come, First Served (FCFS), this permitted new RADIUS commands to be allocated without IETF review. This oversight has since been fixed in "IANA Considerations for RADIUS" [RFC3575].

A.3. TLS Extensions

The Secure Sockets Layer (SSL) v2 protocol was developed by Netscape to be used to secure online transactions on the Internet. It was later replaced by SSL v3, also developed by Netscape. SSL v3 was then further developed by the IETF as the Transport Layer Security (TLS) 1.0 [RFC2246].

The SSL v3 protocol was not explicitly specified to be extended. Even TLS 1.0 did not define an extension mechanism explicitly. However, extension "loopholes" were available. Extension mechanisms were finally defined in "Transport Layer Security (TLS) Extensions" [RFC4366]:

- New versions
- New cipher suites
- Compression
- Expanded handshake messages
- New record types
- New handshake messages

The protocol also defines how implementations should handle unknown extensions.

Of the above extension methods, new versions and expanded handshake messages have caused the most interoperability problems. Implementations are supposed to ignore unknown record types but to reject unknown handshake messages.

The new version support in SSL/TLS includes a capability to define new versions of the protocol, while allowing newer implementations to communicate with older implementations. As part of this functionality, some Key Exchange methods include functionality to prevent version rollback attacks.

The experience with this upgrade functionality in SSL and TLS is decidedly mixed:

- SSL v2 and SSL v3/TLS are not compatible. It is possible to use SSL v2 protocol messages to initiate a SSL v3/TLS connection, but it is not possible to communicate with a SSL v2 implementation using SSL v3/TLS protocol messages.
- There are implementations that refuse to accept handshakes using newer versions of the protocol than they support.
- There are other implementations that accept newer versions, but have implemented the version rollback protection clumsily.

The SSL v2 problem has forced SSL v3 and TLS clients to continue to use SSL v2 Client Hellos for their initial handshake with almost all servers until 2006, much longer than would have been desirable, in order to interoperate with old servers.

The problem with incorrect handling of newer versions has also forced many clients to actually disable the newer protocol versions, either by default, or by automatically disabling the functionality, to be able to connect to such servers. Effectively, this means that the

version rollback protection in SSL and TLS is non-existent if talking to a fatally compromised older version.

SSL v3 and TLS also permitted expansion of the Client Hello and Server Hello handshake messages. This functionality was fully defined by the introduction of TLS Extensions, which makes it possible to add new functionality to the handshake, such as the name of the server the client is connecting to, request certificate status information, indicate Certificate Authority support, maximum record length, etc. Several of these extensions also introduce new handshake messages.

It has turned out that many SSL v3 and TLS implementations that do not support TLS Extensions, did not, as required by the protocol specifications, ignore the unknown extensions, but instead failed to establish connections. Several of the implementations behaving in this manner are used by high profile Internet sites, such as online banking sites, and this has caused a significant delay in the deployment of clients supporting TLS Extensions, and several of the clients that have enabled support are using heuristics that allow them to disable the functionality when they detect a problem.

Looking forward, the protocol version problem, in particular, can cause future security problems for the TLS protocol. The strength of the digest algorithms (MD5 and SHA-1) used by SSL and TLS is weakening. If MD5 and SHA-1 weaken to the point where it is feasible to mount successful attacks against older SSL and TLS versions, the current error recovery used by clients would become a security vulnerability (among many other serious problems for the Internet).

To address this issue, TLS 1.2 [RFC5246] makes use of a newer cryptographic hash algorithm (SHA-256) during the TLS handshake by default. Legacy ciphersuites can still be used to protect application data, but new ciphersuites are specified for data protection as well as for authentication within the TLS handshake. The hashing method can also be negotiated via a Hello extension. Implementations are encouraged to implement new ciphersuites, and to enable the negotiation of the ciphersuite used during a TLS session to be governed by policy, thus enabling a more rapid transition away from weakened ciphersuites.

The lesson to be drawn from this experience is that it isn't sufficient to design extensibility carefully; it must also be implemented carefully by every implementer, without exception. Test suites and certification programs can help provide incentives for implementers to pay attention to implementing extensibility mechanisms correctly.

A.4. L2TP Extensions

Layer Two Tunneling Protocol (L2TP) [RFC2661] carries Attribute-Value Pairs (AVPs), with most AVPs having no semantics to the L2TP protocol itself. However, it should be noted that L2TP message types are identified by a Message Type AVP (Attribute Type 0) with specific AVP values indicating the actual message type. Thus, extensions relating to Message Type AVPs would likely be considered major extensions.

L2TP also provides for Vendor-Specific AVPs. Because everything in L2TP is encoded using AVPs, it would be easy to define vendor-specific AVPs that would be considered major extensions.

L2TP also provides for a "mandatory" bit in AVPs. Recipients of L2TP messages containing AVPs they do not understand but that have the mandatory bit set, are expected to reject the message and terminate the tunnel or session the message refers to. This leads to interesting interoperability issues, because a sender can include a vendor-specific AVP with the M-bit set, which then causes the recipient to not interoperate with the sender. This sort of behavior is counter to the IETF ideals, as implementations of the IETF standard should interoperate successfully with other implementations and not require the implementation of non-IETF extensions in order to interoperate successfully. Section 4.2 of the L2TP specification [RFC2661] includes specific wording on this point, though there was significant debate at the time as to whether such language was by itself sufficient.

Fortunately, it does not appear that the potential problems described above have yet become a problem in practice. At the time of this writing, the authors are not aware of the existence of any vendor-specific AVPs that also set the M-bit.

Change log [RFC Editor: please remove this section]

-14: 2012-6-09. Resolved issue 169.
-13: 2012-6-03. Resolved issue 166.
-12: 2012-5-27. Resolved issues 127, 128, 129, 133 and 161.
-11: 2012-2-22. Resolved issue 126.
-10: 2012-2-12. Resolved issues 106 and 108.
-09: 2011-10-30. Resolved additional issues.
-08: 2011-10-22. Resolved additional issues.
-07: 2011-7-24. Resolved issues raised in Call for Comment.
-06: 2011-3-1. Incorporated corrections and organizational updates.
-05: 2011-2-4. Added to the Security Considerations section.
-04: 2011-2-1. Added material on cryptographic agility.
-03: 2011-1-25. Updates and reorganization.
-02: 2010-7-12. Updates by Bernard Aboba.

-01: 2010-4-7. Updates by Stuart Cheshire.

draft-iab-extension-recs-00: 2009-4-24. Updated boilerplate, author list.

-04: 2008-10-24. Updated author addresses, editorial fixes.

-03: 2008-10-17. Updated references, added material relating to versioning.

-02: 2007-06-15. Reorganized Sections 2 and 3.

-01: 2007-03-04. Updated according to comments, especially the wording about TLS, added various specific examples.

draft-carpenter-extension-recs-00: original version, 2006-10-12. Derived from draft-iesg-vendor-extensions-02.txt dated 2004-06-04 by focusing on architectural issues; the procedural issues were moved to RFC 4775.

Authors' Addresses

Brian Carpenter
Department of Computer Science
University of Auckland
PB 92019
Auckland, 1142
New Zealand

Email: brian.e.carpenter@gmail.com

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

EMail: bernard_aboba@hotmail.com

Stuart Cheshire
Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014

EMail: cheshire@apple.com