Network Working Group
Internet-Draft
Intended Status: Informational
Expires: August 4, 2011

B. Carpenter
B. Aboba (ed)
S. Cheshire
Internet Architecture Board

4 February 2011

Design Considerations for Protocol Extensions
draft-iab-extension-recs-05

Abstract

This document discusses issues related to the extensibility of Internet protocols, with a focus on the architectural design considerations involved. Case study examples are included. It is intended to assist designers of both base protocols and extensions.

Table of Contents

1. Introduction

Internet Engineering Task Force (IETF) protocols typically include mechanisms whereby they can be extended in the future. It is of course a good principle to design extensibility into protocols; one common definition of a successful protocol is one that becomes widely used in ways not originally anticipated, as described in "What Makes for a Successful Protocol" [RFC5218]. Well-designed extensibility mechanisms facilitate the evolution of protocols and help make it easier to roll out incremental changes in an interoperable fashion.

When an initial protocol design is extended, there is always a risk of unintended consequences, such as interoperability problems or security vulnerabilities. This risk 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, extensions should be done carefully and with a full understanding of the base protocol, existing implementations, and current operational practice.

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.

This document describes technical considerations for protocol extensions, in order to minimize such risks. It is intended to assist designers of both base protocols and extensions. Formal procedures for extending IETF protocols are discussed in "Procedures for Protocol Extensions and Variations" BCP 125 [RFC4775].

Section 2 discusses extension documentation and review. 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. Extension Documentation and Review

One of the pre-requisites for interoperable extensibility is proper documentation and review.

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.

To help future extension writers to use extension mechanisms properly, there may be a need for explicit guidance relating to extensions beyond what is encapsulated in the IANA considerations section of the base specification.

Protocols whose data model is likely to be widely extended (particularly using vendor-specific elements) should have a Design Guidelines document specifically addressing extensions. For example, "Guidelines for Authors and Reviewers of MIB Documents" [RFC4181] provides valuable guidance to protocol designers creating new MIB modules.

2.1. When is an Extension Routine?

An extension may be considered 'routine' if it amounts to a new data element of a type that is already supported within the data model, 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, the protocol must have been designed to carry the proposed data type, so that no changes to the underlying base protocol or existing implementations are needed to carry the new data element.

Moreover, no changes should be required to existing and currently deployed implementations of the underlying protocol unless they want to make use of the new data element. Using the existing protocol to carry a new data element should not impact existing implementations or cause operational problems. This typically requires that the protocol silently discard unknown data elements.

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

In order to increase the likelihood that routine extensions are truly routine, protocol documents should provide guidelines explaining how extensions should be performed. 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.

Processes that allow routine extensions with minimal or no review should be used sparingly (such as the "First Come First Served" allocation policy described in "Guidelines for Writing an IANA Considerations Section in RFCs" [RFC5226]). In particular, they should be limited to cases that are unlikely to cause protocol failures, such as allowing new opaque data elements.

2.2. What Constitutes a Major Extension?

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

Extension designers should examine their design for the following issues:

1. Modifications or extensions to the working of the underlying protocol. This can include changing the semantics of existing Protocol Data Units (PDUs) 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 or data types. A base protocol without a "silent discard" rule for unknown data elements may automatically enter this category, even for apparently minor extensions.

2. Changes to the transport model. While there are circumstances where specification of additional transport protocols may make sense, removal of a widely implemented transport protocol is highly likely to result in interoperability problems and thus should be avoided wherever possible.

Where additional transports are specified, one way to avoid issues is to mandate support for a single transport protocol, while designating other transport protocols as optional. However, if optional transport protocols are introduced due to the unique advantages they afford in certain scenarios, in those situations implementations not supporting optional transport protocols may exhibit degraded performance or may even fail.

While requiring support for multiple transport protocols may appear attractive, authors 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.

3. 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.

4. 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.

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 infrastructure require special care.

3.1. Limited Extensibility

Designing a protocol for extensibility may have the perverse side effect of making it easy to construct incompatible extensions. Consequently, protocols should not be made more extensible than clearly necessary at inception, and the process for defining new extensibility mechanisms should ensure that adequate review of proposed extensions will take place before widespread adoption.

3.2. Design for Global Interoperability

The IETF mission [RFC3935] is to create interoperable protocols for the global Internet, not a collection of different incompatible protocols (or "profiles") for use in separate private networks. 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.

As a result, extensions cannot be designed for an isolated environment; instead, extension designers must assume that systems using the extension will need to interoperate with systems 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" extensions encounter each other on a public network, unexpected interoperability problems may occur.

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 "site-specific" or "experimental" use, such as "X-" headers in email messages [RFC0822]. This problem with this approach is that when an experiment turns out to be successful, or a site-specific use turns out to have applicability elsewhere, other vendors will then implement that "X-" header for interoperability, and the "X-" header becomes a de facto standard, meaning that it is no longer true that any header beginning "X-" is site-specific or experimental. The notion of "X-" headers was removed from the Internet Message Format standard when it was was updated in 2001 [RFC2822].

3.3. Architectural Compatibility

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

Documents relying on extension mechanisms need to explicitly identify the mechanisms being relied upon. Where extension guidelines are available, mechanisms need to indicate whether they are compliant with those guidelines and if not, why not. 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.

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".

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], incompatible forks in development can result from the uncoordinated adaptation of a protocol, parameter or code-point. Section 1 of [RFC5704] states:

> 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.

Profiling is a common technique for improving interoperability within a target environment or set of scenarios. Typically, profiles are constructed by narrowing potential implementation choices or by removing protocol features. However, 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 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 also 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.

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 and incorrect extension behavior.

3.6. Protocol Parameter Registration

An extension is often likely to make use of additional values added to an existing IANA registry (in many cases, simply by adding a new Type-Length-Value (TLV) field). To avoid conflicting usage of the same value, as well as to prevent potential difficulties in determining and transferring parameter ownership, it is essential that all new values are properly registered by the applicable procedures.

For general rules see "Guidelines for Writing an IANA Considerations Section in RFCs" [RFC5226], and for specific rules and registries see the individual protocol specification RFCs and the IANA web site. 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. Numerous cases could be cited, but not without embarrassing specific implementers.

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 these policies, if administered clumsily, can have the opposite effect, discouraging protocol parameter registration and encouraging rampant self-allocation. These effects have also been observed in a number of instances.

In some cases, it may be appropriate to use values designated as "experimental" or "local use" in early implementations of an extension. 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.

3.7. Extensions to Critical Infrastructure

Some protocols (such as Domain Name Service (DNS) and Border Gateway Protocol (BGP)) have become critical components of the Internet infrastructure. When such protocols 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.

4. Considerations for the Base Protocol

Good extension design depends on a well designed base protocol. Interoperability stems from a number of factors, including:

1. A well-written specification. Does the specification make clear what an implementor 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 deployability. This includes understanding what current implementations do and how a proposed extension will interact with deployed systems. Will a proposed extension (or its proposed usage) operationally stress existing implementations or the underlying protocol itself if widely deployed?

3. An adequate transition or coexistence story. What impact will the proposed extension have on implementations that do not understand it? Is there a way to negotiate or determine the capabilities of a peer? Can the extended protocol negotiate with an unextended partner to find a common subset of useful functions?

4. Respecting underlying architectural or security assumptions. This includes assumptions that may not be well-documented, those that may have arisen as the result of operational experience, or those that only became understood after the original protocol was published. For example, do the 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?

5. Minimizing impact on critical infrastructure. Does the proposed extension (or its proposed usage) have the potential for negatively impacting critical infrastructure to the point where explicit steps would be appropriate to isolate existing uses from new ones?

6. Data model extensions. Does the proposed extension extend the data model in a major way? For example, are new data types defined that may require code changes 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.

That's bad protocol design and description; it should be clear what the expectation is and how you test it. 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 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 EAP [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 type 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 assumed to be 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 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. An excellent example is the Simple Network Management Protocol (SNMP) SMI. Guidelines exist for defining the 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 finite, 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 rapid parameter space exhaustion. 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.

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. Where vendor-specific parameter support is available, this may be achieved by allocating an 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 against a bidding-down attack while retaining backward compatibility.

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

[I-D.ietf-radext-design]
DeKok, A. and G. Weber, "RADIUS Design Guidelines", draft-ietf-radext-design-19.txt, Internet draft (work in

progress), November 2010.

[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.

[RFC0822] 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.

[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.

[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.

[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.

[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.

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

[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.

[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", RFC 5727, March 2010.

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 Jari Arkko, Ted Hardie, Loa Andersson, Eric Rescorla, Pekka Savola, Leslie Daigle, Alfred Hoenes, Adam Roach and Phillip Hallam-Baker.

The text on TLS experience was contributed by Yngve Pettersen.

IAB Members at the Time of this Writing

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]

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.

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. Since the VSA mechanism was not designed for interoperability, VSAs do not contain a "mandatory" bit. As a result, RADIUS clients and servers may not know whether it is safe to ignore unknown attributes. For example, Section 5 of the RADIUS specification [RFC2865] states:

> A RADIUS server MAY ignore Attributes with an unknown Type. A RADIUS client MAY ignore Attributes with an unknown Type.

However, in the case where the VSAs pertain to security (e.g. Filters) it may not be safe to ignore them, since the RADIUS specification [RFC2865] also states:

> A NAS that does not implement a given service MUST NOT implement the RADIUS attributes for that service. For example, a NAS that is unable to offer ARAP service MUST NOT implement the RADIUS attributes for ARAP. A NAS MUST treat a RADIUS access-accept authorizing an unavailable service as an access-reject instead."

Detailed discussion of the issues arising from this can be found in "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes" [RFC5080] Section 2.5.

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. However, 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" [I-D.ietf-radext-design].

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 essentially enabled 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]

draft-iab-extension-recs-04: 2011-2-3. Added a section on cryptographic agility. Additional reorganization.

draft-iab-extension-recs-03: 2011-1-25. Updates and reorganization to reflect comments from the IETF community.

draft-iab-extension-recs-02: 2010-7-12. Updates by Bernard Aboba

draft-iab-extension-recs-01: 2010-4-7. Updates by Stuart Cheshire.

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

draft-carpenter-extension-recs-04: 2008-10-24. Updated author addresses, fixed editorial issues.

draft-carpenter-extension-recs-03: 2008-10-17. Updated references, added material relating to versioning.

draft-carpenter-extension-recs-02: 2007-06-15. Reorganized Sections 2 and 3.

draft-carpenter-extension-recs-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 more procedural issues in that draft 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