Network Working Group
Internet-Draft
Intended Status: Informational
Expires: December 25, 2007

B. Carpenter (ed)
IBM
B. Aboba
Microsoft Corporation
12 June 2007

Design Considerations for Protocol Extensions
draft-carpenter-extension-recs-02

Copyright Notice

Abstract

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

Table of Contents

1. Introduction

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. 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 300 existing RFCs. 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 BCP 125 [RFC4775].

Section 2 describes architectural principles for protocol extensibility. Section 3 is aimed principally at extension designers, and Section 4 at base protocol designers. Nevertheless, readers are advised to study the whole document, since the considerations are closely linked.

2. Architectural Principles

This Section describes basic principles of protocol extensibility:

1. Extensibility features should be limited to what is clearly necessary when the protocol is developed.

2. Protocol extensions should be designed for global interoperability.

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

4. Extension mechanisms need to be fully documented.

5. Extension mechanisms need to be testable.

6. Protocol parameters should be registered and used for their intended purpose.

7. Extensions to critical infrastructure should not impact the security or reliability of the global Internet.

8. Extension mechanisms should be explicitly identified and should be architecturally compatible with the base protocol design.

2.1. Limited Extensibility

Protocols that permit easy extensions 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. In practice, this means First Come First Served [RFC2434] and similar policies that allow routine extensions should be used sparingly, as they imply minimal or no review. In particular, they should be limited to cases, such as allowing new opaque data elements, that are unlikely to cause protocol failures.

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

they 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.

2.2. Global Interoperability

Global interoperability is a primary goal of IETF protocol design. Experience shows that software is often used outside the particular special environment it was originally intended for, so extensions cannot be designed for an isolated environment. Designers of extensions must assume the high likelihood of a system using the extension having to interoperate with systems on the global Internet.

For this reason, 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.

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

2.3. Protocol Variations

Protocol extension mechanisms should not be used to create incompatible forks in development instead. 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.

An example of what might go wrong is misuse of the X- mail headers originally defined in SMTP [RFC0822]. X-anything was a valid mail header; but it had no defined meaning in the standard. Suppose a mail implementation assigns specific semantics to X-anything that causes it to take specific action, such as discarding a message as spam. If it encounters a message from a different implementation that assigns different semantics, it may take quite inappropriate action, such as discarding a valid message. Thus, relying on the

implied semantics of an X- header automatically creates a risk of operational failures. X- headers were removed from [RFC2822]. Even when they are assigned semantics, as in [RFC4356], great care must be taken that implementations do not rely on such semantics in messages that have crossed the open Internet.

Thus we observe that 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.

Protocol variations - specifications that look very similar to the original but are actually different - are even more harmful to interoperability than extensions. In general, such variations should be avoided. If they cannot be avoided, as many of the following considerations as possible should be applied, to minimize the damage to interoperability.

2.4. Extension Documentation

Some protocol components are designed with the specific intention of allowing extensibility. These should be clearly identified, with specific and complete instructions on how to extend them, including the process for adequate review of extension proposals: do they need community review and if so how much and by whom? For example, the definition of additional data elements that can be carried opaquely

may require no review, while the addition of new data types or new protocol messages might require a Standards Track action. Guidance on writing appropriate IANA Considerations text may be found in [RFC2434].

In a number of cases, there is a need for explicit guidance relating to extensions beyond what is encapsulated in the IANA considerations section of the base specification. The usefulness of [RFC4181] would appear to suggest that protocols whose data model is likely to be widely extended (particularly using vendor-specific elements) need a Design Guidelines document specifically addressing extensions.

2.5. Testability

Experience shows that it is insufficient 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 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 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.

2.6. 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 "TLV" (type-length-value) field). To avoid conflicting usage of the same value, it is essential that all new values are properly registered by the applicable procedures. See BCP 26, [RFC2434] for the general rules, and individual protocol RFCs, and the IANA web site, for specific rules and registries. 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 TCP (or 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 implementors.

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

extension. For example, [RFC4727] discusses experimental values for IP and transport headers, and [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.

2.7. Extensions to Critical Infrastructure

Some protocols (such as DNS and 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.

2.8. Architectural Compatibility

Since protocol extension mechanisms may impact interoperability, it is important that these mechanisms be architecturally compatible with the base protocol. This implies that documents relying on extension mechanisms need to explicitly identify them, rather than burying them in the text in the hope that they will escape notice.

As part of the definition of new extension mechanisms, the authors need 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.

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. Specific Considerations for Robust Extensions

This section makes explicit some design considerations based on the community's experience with extensibility mechanisms.

3.1. Interoperability Checklist

Good interoperability stems from a number of factors, including:

1. Having 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. Learning lessons from deployment. 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. Having 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?

3.2. 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 are 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 DHC vendor-specific option, RADIUS Vendor-Specific Attributes compliant with [RFC2865], the enterprise OID tree for MIB modules, vendor MIME types, and some classes of (non-critical) certification extensions. Such extensions can safely be made with minimal discussion.

3.3. 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 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 basic architectural assumptions. This includes 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.

4. Considerations for the Base Protocol

A good extension design depends on a good base protocol. Ideally, the document that defines a base protocol's extension mechanisms will include guidance to future extension writers that help them use extension mechanisms properly. It may also be possible to define classes of extensions that need little or no review, while other classes need wide review. The details will necessarily be technology-specific.

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 has shown that these profiles have limitations, so the

ROHC WG is developing ROHCv2. A ROHCv1 implementation will not contain code for the ROHCv2 profiles. As the ROHC WG charter said at the time of writing:

> 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 EAP [RFC3748] as well as 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. In this approach, implementations exchange the highest supported version, 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 higher version number implies support for lower versions. Where these assumptions are invalid, this approach breaks down, potentially resulting in interoperability problems. An example of this issue occurs in [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 [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.

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). If this is not done, future implementation of new values in the reserved field may break old software. Similarly, protocols should carefully specify how receivers should react to unknown TLVs etc., such that failures occur only when that is truly the desired result.

4.3. Encoding Formats

Using widely-supported encoding formats leads to better interoperability and easier extensibility. An excellent example is the 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.

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).

6. IANA Considerations

This draft requires no action by IANA.

7. 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, Bernard Aboba 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, and Leslie Daigle.

The text on TLS experience was contributed by Yngve Pettersen.

8. References

8.1. Normative References

[RFC0822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

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

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 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.

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

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

[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.

[RFC3546] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

[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.

[RFC4356] Gellens, R., "Mapping Between the Multimedia Messaging Service (MMS) and Internet Mail", RFC 4356, January 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.

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

8.2. Informative References

[I-D.andersson-rtg-gmpls-change] Andersson, L. and A. Farrel, "Change Process for Multiprotocol Label Switching (MPLS) and Generalized MPLS (GMPLS) Protocols and Procedures", draft-andersson-rtg-gmpls-change-08 (work in progress), March 2007.

[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, Internet draft (work in progress), October 2004.

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

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

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

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

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

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]
The (G)MPLS change process (mainly procedural)
[I-D.andersson-rtg-gmpls-change]
LDAP extensions[RFC4521]
EPP extensions[RFC3735]
DNS extensions[RFC2671]

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.

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

> 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 SDOs to define their 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, [RFC2865] Section 5 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 [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."

Since it was not envisaged that multi-vendor VSA implementations would need to interoperate, [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.

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 [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 [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) protocol.

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 [RFC3546]:

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

- o 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.
- o There are implementations that refuse to accept handshakes using newer versions of the protocol than they support.
- o 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 introduces new handshake messages.

It has turned out that many SSL v3 and TLS implementations that do not support TLS Extensions, did not, as specified in the protocols,

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 and TLS is weakening, and work is underway to define TLS 1.2 which will permit new methods to be used in the protocol instead of the currently used methods. 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).

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.

A.4. L2TP Extensions

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 cause 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 above concerns have been a problem in practice. At the time of this writing, the authors are unaware of the existence of vendor-specific AVPs that also set the M-bit.

Change log [RFC Editor: please remove this section]

draft-carpenter-extension-rec-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
IBM
8 Chemin de Blandonnet
1214 Vernier,
Switzerland

Email: brian.e.carpenter@gmail.com

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

EMail: bernarda@microsoft.com
Phone: +1 425 706 6605
Fax: +1 425 936 7329

Full Copyright Statement

OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).