Internet Engineering Task Force (IETF)
Request for Comments: 6929
Updates: 2865, 3575, 6158
Category: Standards Track
ISSN: 2070-1721

A. DeKok
Network RADIUS
A. Lior

April 2013

Remote Authentication Dial-In User Service (RADIUS) Protocol Extensions

Abstract

The Remote Authentication Dial-In User Service (RADIUS) protocol is nearing exhaustion of its current 8-bit Attribute Type space. In addition, experience shows a growing need for complex grouping, along with attributes that can carry more than 253 octets of data. This document defines changes to RADIUS that address all of the above problems.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6929.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction

Under current allocation pressure, we expect that the RADIUS Attribute Type space will be exhausted by 2014 or 2015. We therefore need a way to extend the type space so that new specifications may continue to be developed. Other issues have also been shown with RADIUS. The attribute grouping method defined in [RFC2868] has been shown to be impractical, and a more powerful mechanism is needed. Multiple Attributes have been defined that transport more than the 253 octets of data originally envisioned with the protocol. Each of these attributes is handled as a "special case" inside of RADIUS implementations, instead of as a general method. We therefore also

need a standardized method of transporting large quantities of data. Finally, some vendors are close to allocating all of the Attributes within their Vendor-Specific Attribute space. It would be useful to leverage changes to the base protocol for extending the Vendor-Specific Attribute space.

We satisfy all of these requirements through the following changes given in this document:

* Defining an "Extended Type" format, which adds 8 bits of "Extended Type" to the RADIUS Attribute Type space, by using one octet of the "Value" field. This method gives us a general way of extending the Attribute Type space (Section 2.1).

* Allocating 4 attributes as using the format of "Extended Type". This allocation extends the RADIUS Attribute Type space by approximately 1000 values (Sections 3.1, 3.2, 3.3, and 3.4).

* Defining a "Long Extended Type" format, which inserts an additional octet between the "Extended Type" octet and the "Value" field. This method gives us a general way of adding more functionality to the protocol (Section 2.2).

* Defining a method that uses the additional octet in the "Long Extended Type" to indicate data fragmentation across multiple Attributes. This method provides a standard way for an Attribute to carry more than 253 octets of data (Section 2.2).

* Allocating 2 attributes as using the format "Long Extended Type". This allocation extends the RADIUS Attribute Type space by an additional 500 values (Sections 3.5 and 3.6).

* Defining a new "Type-Length-Value" (TLV) data type. This data type allows an attribute to carry TLVs as "sub-Attributes", which can in turn encapsulate other TLVs as "sub-sub-Attributes". This change creates a standard way to group a set of Attributes (Section 2.3).

* Defining a new "Extended-Vendor-Specific" (EVS) data type. This data type allows an attribute to carry Vendor-Specific Attributes (VSAs) inside of the new Attribute formats (Section 2.4).

* Defining a new "integer64" data type. This data type allows counters that track more than 2^32 octets of data (Section 2.5).

* Allocating 6 attributes using the new EVS data type. This allocation extends the Vendor-Specific Attribute Type space by over 1500 values (Sections 4.1 through 4.6).

* Defining the "Vendor-Id" for Vendor-Specific Attributes to encompass the entire 4 octets of the Vendor field. [RFC2865] Section 5.26 defined it to be 3 octets, with the fourth octet being zero (Section 2.6).

* Describing compatibility with existing RADIUS systems (Section 5).

* Defining guidelines for the use of these changes for IANA, implementations of this specification, and for future RADIUS specifications (Section 6).

As with any protocol change, the changes defined here are the result of a series of compromises. We have tried to find a balance between flexibility, space in the RADIUS message, compatibility with existing deployments, and difficulty of implementation.

1.1. Caveats and Limitations

This section describes some caveats and limitations of the proposal.

1.1.1. Failure to Meet Certain Goals

One goal that was not met by the above modifications is to have an incentive for standards to use the new space. That incentive is being provided by the exhaustion of the standard space.

1.1.2. Implementation Recommendations

It is RECOMMENDED that implementations support this specification. It is RECOMMENDED that new specifications use the formats defined in this specification.

The alternative to the above recommendations is a circular argument of not implementing this specification because no other standards reference it, and also not defining new standards referencing this specification because no implementations exist.

As noted earlier, the standard space is almost entirely allocated. Ignoring the looming crisis benefits no one.

1.2. Terminology

This document uses the following terms:

Silently discard

This means the implementation discards the packet without further processing. The implementation MAY provide the capability of logging the error, including the contents of the silently discarded packet, and SHOULD record the event in a statistics counter.

Invalid attribute

This means that the Length field of an Attribute is valid (as per [RFC2865], Section 5, top of page 25) but the contents of the Attribute do not follow the correct format, for example, an Attribute of type "address" that encapsulates more than four, or less than four, octets of data. See Section 2.8 for a more complete definition.

Standard space

This refers to codes in the RADIUS Attribute Type space that are allocated by IANA and that follow the format defined in Section 5 of [RFC2865].

Extended space

This refers to codes in the RADIUS Attribute Type space that require the extensions defined in this document and are an extension of the standard space, but that cannot be represented within the standard space.

Short extended space

This refers to codes in the extended space that use the "Extended Type" format.

Long extended space

This refers to codes in the extended space that use the "Long Extended Type" format.

The following terms are used here with the meanings defined in BCP 26 [RFC5226]: "namespace", "assigned value", "registration", "Private Use", "Reserved", "Unassigned", "IETF Review", and "Standards Action".

1.3. Requirements Language

In this document, several words are used to signify the requirements of the specification. 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 [RFC2119].

2. Extensions to RADIUS

This section defines two new Attribute formats: "Extended Type" and "Long Extended Type". It defines a new Type-Length-Value (TLV) data type, an Extended-Vendor-Specific (EVS) data type, and an Integer64 data type. It defines a new method for naming attributes and identifying Attributes using the new Attribute formats. It finally defines the new term "invalid attribute" and describes how it affects implementations.

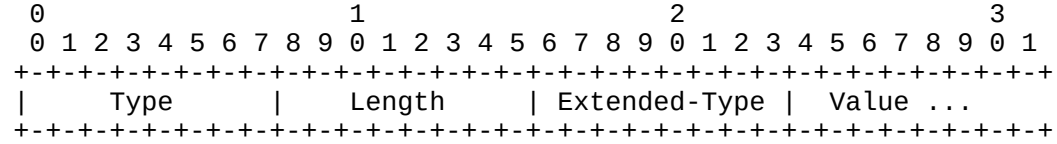
The new Attribute formats are designed to be compatible with the Attribute format given in [RFC2865] Section 5. The meaning and interpretation of the Type and Length fields are unchanged from that specification. This reuse allows the new formats to be compatible with RADIUS implementations that do not implement this specification. Those implementations can simply ignore the "Value" field of an attribute or forward it verbatim.

The changes to the Attribute format come about by "stealing" one or more octets from the "Value" field. This change has the effect that the "Value" field of [RFC2865] Section 5 contains both the new octets given here and any attribute-specific Value. The result is that "Value"s in this specification are limited to less than 253 octets in size. This limitation is overcome through the use of the "Long Extended Type" format.

We reiterate that the formats given in this document do not insert new data into an attribute. Instead, we "steal" one octet of Value, so that the definition of the Length field remains unchanged. The new Attribute formats are designed to be compatible with the Attribute format given in [RFC2865] Section 5. The meaning and interpretation of the Type and Length fields is unchanged from that specification. This reuse allows the new formats to be compatible with RADIUS implementations that do not implement this specification. Those implementations can simply ignore the "Value" field of an attribute or forward it verbatim.

2.1. Extended Type

This section defines a new Attribute format, called "Extended Type". A summary of the Attribute format is shown below. The fields are transmitted from left to right.



Type

This field is identical to the Type field of the Attribute format defined in [RFC2865] Section 5.

Length

The Length field is one octet and indicates the length of this Attribute, including the Type, Length, "Extended-Type", and "Value" fields. Permitted values are between 4 and 255. If a client or server receives an Extended Attribute with a Length of 2 or 3, then that Attribute MUST be considered to be an "invalid attribute" and handled as per Section 2.8, below.

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified according to the policies and rules described in Section 10. Unlike the Type field defined in [RFC2865] Section 5, no values are allocated for experimental or implementation-specific use. Values 241-255 are reserved and MUST NOT be used.

The Extended-Type is meaningful only within a context defined by the Type field. That is, this field may be thought of as defining a new type space of the form "Type.Extended-Type". See Section 3.5, below, for additional discussion.

A RADIUS server MAY ignore Attributes with an unknown "Type.Extended-Type".

A RADIUS client MAY ignore Attributes with an unknown "Type.Extended-Type".

Value

This field is similar to the "Value" field of the Attribute format defined in [RFC2865] Section 5. The format of the data MUST be a valid RADIUS data type.

The "Value" field is one or more octets.

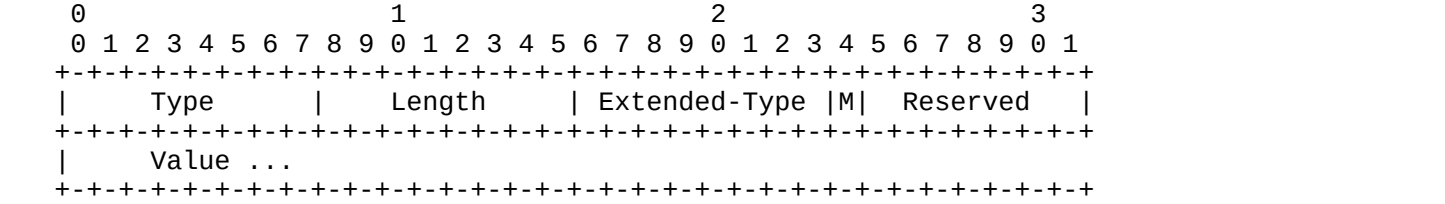
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type" to determine the interpretation of the "Value" field.

The addition of the Extended-Type field decreases the maximum length for attributes of type "text" or "string" from 253 to 252 octets. Where an Attribute needs to carry more than 252 octets of data, the "Long Extended Type" format MUST be used.

Experience has shown that the "experimental" and "implementation-specific" attributes defined in [RFC2865] Section 5 have had little practical value. We therefore do not continue that practice here with the Extended-Type field.

2.2. Long Extended Type

This section defines a new Attribute format, called "Long Extended Type". It leverages the "Extended Type" format in order to permit the transport of attributes encapsulating more than 253 octets of data. A summary of the Attribute format is shown below. The fields are transmitted from left to right.



Type

This field is identical to the Type field of the Attribute format defined in [RFC2865] Section 5.

Length

The Length field is one octet and indicates the length of this Attribute, including the Type, Length, Extended-Type, and "Value" fields. Permitted values are between 5 and 255. If a client or

server receives a "Long Extended Type" with a Length of 2, 3, or 4, then that Attribute MUST be considered to be an "invalid attribute" and handled as per Section 2.8, below.

Note that this Length is limited to the length of this fragment. There is no field that gives an explicit value for the total size of the fragmented attribute.

Extended-Type

This field is identical to the Extended-Type field defined above in Section 2.1.

M (More)

The More field is one (1) bit in length and indicates whether or not the current attribute contains "more" than 251 octets of data. The More field MUST be clear (0) if the Length field has a value of less than 255. The More field MAY be set (1) if the Length field has a value of 255.

If the More field is set (1), it indicates that the "Value" field has been fragmented across multiple RADIUS attributes. When the More field is set (1), the Attribute MUST have a Length field of value 255, there MUST be an attribute following this one, and the next attribute MUST have both the same Type and "Extended Type". That is, multiple fragments of the same value MUST be in order and MUST be consecutive attributes in the packet, and the last attribute in a packet MUST NOT have the More field set (1).

That is, a packet containing a fragmented attribute needs to contain all fragments of the Attribute, and those fragments need to be contiguous in the packet. RADIUS does not support inter-packet fragmentation, which means that fragmenting an attribute across multiple packets is impossible.

If a client or server receives an attribute fragment with the "More" field set (1) but for which no subsequent fragment can be found, then the fragmented attribute is considered to be an "invalid attribute" and handled as per Section 2.8, below.

Reserved

This field is 7 bits long and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Future specifications may define additional meaning for this field. Implementations therefore MUST NOT treat this field as invalid if it is non-zero.

Value

This field is similar to the "Value" field of the Attribute format defined in [RFC2865] Section 5. It may contain a complete set of data (when the Length field has a value of less than 255), or it may contain a fragment of data.

The "Value" field is one or more octets.

Implementations supporting this specification MUST use the identifier of "Type.Extended-Type" to determine the interpretation of the "Value" field.

Any interpretation of the resulting data MUST occur after the fragments have been reassembled. The length of the data MUST be taken as the sum of the lengths of the fragments (i.e., "Value" fields) from which it is constructed. The format of the data SHOULD be a valid RADIUS data type. If the reassembled data does not match the expected format, all fragments MUST be treated as "invalid attributes", and the reassembled data MUST be discarded.

We note that the maximum size of a fragmented attribute is limited only by the RADIUS packet length limitation (i.e., 4096 octets, not counting various headers and overhead). Implementations MUST be able to handle the case where one fragmented attribute completely fills the packet.

This definition increases the RADIUS Attribute Type space as above but also provides for transport of Attributes that could contain more than 253 octets of data.

Note that [RFC2865] Section 5 says:

If multiple Attributes with the same Type are present, the order of Attributes with the same Type MUST be preserved by any proxies. The order of Attributes of different Types is not required to be preserved. A RADIUS server or client MUST NOT have any dependencies on the order of attributes of different types. A RADIUS server or client MUST NOT require attributes of the same type to be contiguous.

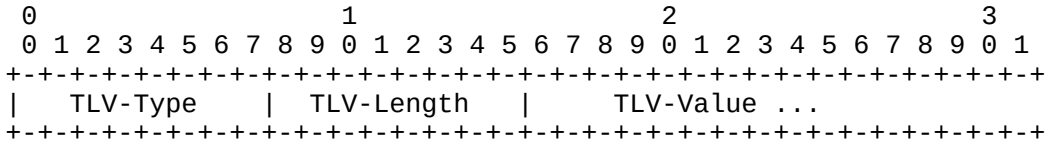
These requirements also apply to the "Long Extended Type" Attribute, including fragments. Implementations MUST be able to process non-contiguous fragments -- that is, fragments that are mixed

together with other attributes of a different Type. This will allow them to accept packets, so long as the Attributes can be correctly decoded.

2.3. TLV Data Type

We define a new data type in RADIUS, called "tlv". The "tlv" data type is an encapsulation layer that permits the "Value" field of an Attribute to contain new sub-Attributes. These sub-Attributes can in turn contain "Value"s of data type TLV. This capability both extends the Attribute space and permits "nested" attributes to be used. This nesting can be used to encapsulate or group data into one or more logical containers.

The "tlv" data type reuses the RADIUS Attribute format, as given below:



TLV-Type

The TLV-Type field is one octet. Up-to-date values of this field are specified according to the policies and rules described in Section 10. Values 254-255 are "Reserved" for use by future extensions to RADIUS. The value 26 has no special meaning and MUST NOT be treated as a Vendor-Specific Attribute.

As with the Extended-Type field defined above, the TLV-Type is meaningful only within the context defined by "Type" fields of the encapsulating Attributes. That is, the field may be thought of as defining a new type space of the form "Type.Extended-Type.TLV-Type". Where TLVs are nested, the type space is of the form "Type.Extended-Type.TLV-Type.TLV-Type", etc.

A RADIUS server MAY ignore Attributes with an unknown "TLV-Type".

A RADIUS client MAY ignore Attributes with an unknown "TLV-Type".

A RADIUS proxy SHOULD forward Attributes with an unknown "TLV-Type" verbatim.

TLV-Length

The TLV-Length field is one octet and indicates the length of this TLV, including the TLV-Type, TLV-Length, and TLV-Value fields. It MUST have a value between 3 and 255. If a client or server receives a TLV with an invalid TLV-Length, then the Attribute that encapsulates that TLV MUST be considered to be an "invalid attribute" and handled as per Section 2.8, below.

TLV-Value

The TLV-Value field is one or more octets and contains information specific to the Attribute. The format and length of the TLV-Value field are determined by the TLV-Type and TLV-Length fields.

The TLV-Value field SHOULD encapsulate a standard RADIUS data type. Non-standard data types SHOULD NOT be used within TLV-Value fields. We note that the TLV-Value field MAY also contain one or more attributes of data type TLV; data type TLV allows for simple grouping and multiple layers of nesting.

The TLV-Value field is limited to containing 253 or fewer octets of data. Specifications that require a TLV to contain more than 253 octets of data are incompatible with RADIUS and need to be redesigned. Specifications that require the transport of empty "Value"s (i.e., Length = 2) are incompatible with RADIUS and need to be redesigned.

The TLV-Value field MUST NOT contain data using the "Extended Type" formats defined in this document. The base Extended Attributes format allows for sufficient flexibility that nesting them inside of a TLV offers little additional value.

This TLV definition is compatible with the suggested format of the "String" field of the Vendor-Specific Attribute, as defined in [RFC2865] Section 5.26, though that specification does not discuss nesting.

Vendors MAY use attributes of type "TLV" in any Vendor-Specific Attribute. It is RECOMMENDED to use type "TLV" for VSAs, in preference to any other format.

If multiple TLVs with the same TLV-Type are present, the order of TLVs with the same TLV-Type MUST be preserved by any proxies. The order of TLVs of different TLV-Types is not required to be preserved. A RADIUS server or client MUST NOT have any dependencies on the order of TLVs of different TLV-Types. A RADIUS server or client MUST NOT require TLVs of the same TLV-Type to be contiguous.

The interpretation of multiple TLVs of the same TLV-Type MUST be that of a logical "and", unless otherwise specified. That is, multiple TLVs are interpreted as specifying an unordered set of values. Specifications SHOULD NOT define TLVs to be interpreted as a logical "or". Doing so would mean that a RADIUS client or server would make an arbitrary and non-deterministic choice among the values.

2.3.1. TLV Nesting

TLVs may contain other TLVs. When this occurs, the "container" TLV MUST be completely filled by the "contained" TLVs. That is, the "container" TLV-Length field MUST be exactly two (2) more than the sum of the "contained" TLV-Length fields. If the "contained" TLVs overfill the "container" TLV, the "container" TLV MUST be considered to be an "invalid attribute" and handled as described in Section 2.8, below.

The depth of TLV nesting is limited only by the restrictions on the TLV-Length field. The limit of 253 octets of data results in a limit of 126 levels of nesting. However, nesting depths of more than 4 are NOT RECOMMENDED. They have not been demonstrated to be necessary in practice, and they appear to make implementations more complex. Reception of packets with such deeply nested TLVs may indicate implementation errors or deliberate attacks. Where implementations do not support deep nesting of TLVs, it is RECOMMENDED that the unsupported layers are treated as "invalid attributes".

2.4. EVS Data Type

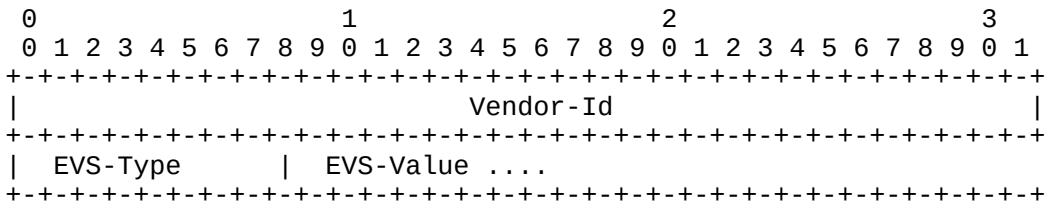
We define a new data type in RADIUS, called "evs", for "Extended-Vendor-Specific". The "evs" data type is an encapsulation layer that permits the EVS-Value field of an Attribute to contain a Vendor-Id, followed by an EVS-Type, and then vendor-defined data. This data can in turn contain valid RADIUS data types or any other data as determined by the vendor.

This data type is intended for use in attributes that carry vendor-specific information, as is done with the Vendor-Specific Attribute (Attribute number 26). It is RECOMMENDED that this data type be used by a vendor only when the Vendor-Specific Attribute Type space has been fully allocated.

Where [RFC2865] Section 5.26 makes a recommendation for the format of the data following the Vendor-Id, we give a strict definition. Experience has shown that many vendors have not followed the [RFC2865] recommendations, leading to interoperability issues. We hope here to give vendors sufficient flexibility as to meet their needs while minimizing the use of non-standard VSA formats.

The "evs" data type MAY be used in Attributes having the format of "Extended Type" or "Long Extended Type". It MUST NOT be used in any other Attribute definition, including standard RADIUS attributes, TLVs, and VSAs.

A summary of the "evs" data type format is shown below. The fields are transmitted from left to right.



Vendor-Id

The 4 octets of the Vendor-Id field are the Network Management Private Enterprise Code [PEN] of the vendor in network byte order.

EVS-Type

The EVS-Type field is one octet. Values are assigned at the sole discretion of the vendor.

EVS-Value

The EVS-Value field is one or more octets. It SHOULD encapsulate a standard RADIUS data type. Using non-standard data types is NOT RECOMMENDED. We note that the EVS-Value field may be of data type TLV. However, it MUST NOT be of data type "evs", as the use cases are unclear for one vendor delegating Attribute Type space to another vendor.

The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets. While we recognize that vendors have complete control over the contents and format of the EVS-Value field, we recommend that good practices be followed.

Further codification of the range of allowed usage of this field is outside the scope of this specification.

Note that unlike the format described in [RFC2865] Section 5.26, this data type has no "Vendor-Length" field. The length of the EVS-Value field is implicit and is determined by taking the "Length" of the encapsulating RADIUS attribute and then subtracting the length of the

Attribute header (2 octets), the "Extended Type" (1 octet), the Vendor-Id (4 octets), and the EVS-Type (1 octet). That is, for "Extended Type" Attributes the length of the EVS-Value field is eight (8) less than the value of the Length field, and for "Long Extended Type" Attributes the length of the EVS-Value field is nine (9) less than the value of the Length field.

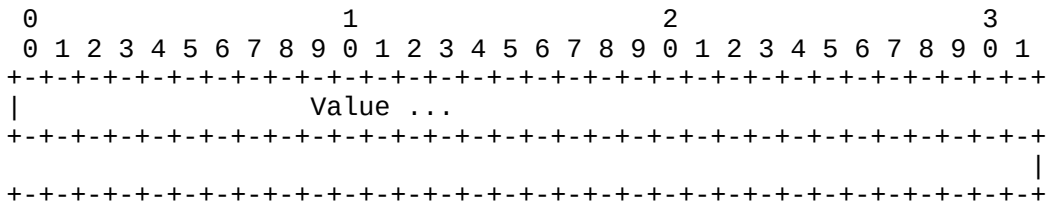
2.5. Integer64 Data Type

We define a new data type in RADIUS, called "integer64", which carries a 64-bit unsigned integer in network byte order.

This data type is intended to be used in any situation where there is a need to have counters that can count past 2^32. The expected use of this data type is within Accounting-Request packets, but this data type SHOULD be used in any packet where 32-bit integers are expected to be insufficient.

The "integer64" data type can be used in Attributes of any format, standard space, extended attributes, TLVs, and VSAs.

A summary of the "integer64" data type format is shown below. The fields are transmitted from left to right.



Attributes having data type "integer64" MUST have the relevant Length field set to eight more than the length of the Attribute header. For standard space Attributes and TLVs, this means that the Length field MUST be set to ten (10). For "Extended Type" Attributes, the Length field MUST be set to eleven (11). For "Long Extended Type" Attributes, the Length field MUST be set to twelve (12).

2.6. Vendor-Id Field

We define the Vendor-Id field of Vendor-Specific Attributes to encompass the entire 4 octets of the Vendor field. [RFC2865] Section 5.26 defined it to be 3 octets, with the fourth octet being zero. This change has no immediate impact on RADIUS, as the maximum Private Enterprise Code defined is still within 16 bits.

However, it is best to make advance preparations for changes in the protocol. As such, it is RECOMMENDED that all implementations support four (4) octets for the Vendor-Id field, instead of three (3).

2.7. Attribute Naming and Type Identifiers

Attributes have traditionally been identified by a unique name and number. For example, the Attribute "User-Name" has been allocated number one (1). This scheme needs to be extended in order to be able to refer to attributes of "Extended Type", and to TLVs. It will also be used by IANA for allocating RADIUS Attribute Type values.

The names and identifiers given here are intended to be used only in specifications. The system presented here may not be useful when referring to the contents of a RADIUS packet. It imposes no requirements on implementations, as implementations are free to reference RADIUS attributes via any method they choose.

2.7.1. Attribute and TLV Naming

RADIUS specifications traditionally use names consisting of one or more words, separated by hyphens, e.g., "User-Name". However, these names are not allocated from a registry, and there is no restriction other than convention on their global uniqueness.

Similarly, vendors have often used their company name as the prefix for VSA names, though this practice is not universal. For example, for a vendor named "Example", the name "Example-Attribute-Name" SHOULD be used instead of "Attribute-Name". The second form can conflict with attributes from other vendors, whereas the first form cannot.

It is therefore RECOMMENDED that specifications give names to Attributes that attempt to be globally unique across all RADIUS Attributes. It is RECOMMENDED that a vendor use its name as a unique prefix for attribute names, e.g., Livingston-IP-Pool instead of IP-Pool. It is RECOMMENDED that implementations enforce uniqueness on names; not doing so would lead to ambiguity and problems.

We recognize that these suggestions may sometimes be difficult to implement in practice.

TLVs SHOULD be named with a unique prefix that is shared among related attributes. For example, a specification that defines a set of TLVs related to time could create attributes called "Time-Zone", "Time-Day", "Time-Hour", "Time-Minute", etc.

2.7.2. Attribute Type Identifiers

The RADIUS Attribute Type space defines a context for a particular "Extended-Type" field. The "Extended-Type" field allows for 256 possible type code values, with values 1 through 240 available for allocation. We define here an identification method that uses a "dotted number" notation similar to that used for Object Identifiers (OIDs), formatted as "Type.Extended-Type".

For example, an attribute within the Type space of 241, having Extended-Type of one (1), is uniquely identified as "241.1". Similarly, an attribute within the Type space of 246, having Extended-Type of ten (10), is uniquely identified as "246.10".

2.7.3. TLV Identifiers

We can extend the Attribute reference scheme defined above for TLVs. This is done by leveraging the "dotted number" notation. As above, we define an additional TLV Type space, within the "Extended Type" space, by appending another "dotted number" in order to identify the TLV. This method can be repeated in sequence for nested TLVs.

For example, let us say that "245.1" identifies RADIUS Attribute Type 245, containing an "Extended Type" of one (1), which is of type "TLV". That attribute will contain 256 possible TLVs, one for each value of the TLV-Type field. The first TLV-Type value of one (1) can then be identified by appending a ".1" to the number of the encapsulating attribute ("241.1"), to yield "241.1.1". Similarly, the sequence "245.2.3.4" identifies RADIUS attribute 245, containing an "Extended Type" of two (2), which is of type "TLV", which in turn contains a TLV with TLV-Type number three (3), which in turn contains another TLV, with TLV-Type number four (4).

2.7.4. VSA Identifiers

There has historically been no method for numerically addressing VSAs. The "dotted number" method defined here can also be leveraged to create such an addressing scheme. However, as the VSAs are completely under the control of each individual vendor, this section provides a suggested practice but does not define a standard of any kind.

The Vendor-Specific Attribute has been assigned the Attribute number 26. It in turn carries a 32-bit Vendor-Id, and possibly additional VSAs. Where the VSAs follow the format recommended by [RFC2865] Section 5.26, a VSA can be identified as "26.Vendor-Id.Vendor-Type".

For example, Livingston has Vendor-Id 307 and has defined an attribute "IP-Pool" as number 6. This VSA can be uniquely identified as 26.307.6, but it cannot be uniquely identified by name, as other vendors may have used the same name.

Note that there are few restrictions on the size of the numerical values in this notation. The Vendor-Id is a 32-bit number, and the VSA may have been assigned from a 16-bit Vendor-Specific Attribute Type space. Implementations SHOULD be capable of handling 32-bit numbers at each level of the "dotted number" notation.

For example, the company USR has historically used Vendor-Id 429 and has defined a "Version-Id" attribute as number 32768. This VSA can be uniquely identified as 26.429.32768 but again cannot be uniquely identified by name.

Where a VSA is a TLV, the "dotted number" notation can be used as above: 26.Vendor-Id.Vendor-Type.TLV1.TLV2.TLV3, where the "TLVn" values are the numerical values assigned by the vendor to the different nested TLVs.

2.8. Invalid Attributes

The term "invalid attribute" is new to this specification. It is defined to mean that the Length field of an Attribute permits the packet to be accepted as not being "malformed". However, the "Value" field of the Attribute does not follow the format required by the data type defined for that Attribute, and therefore the Attribute is "malformed". In order to distinguish the two cases, we refer to "malformed" packets and "invalid attributes".

For example, an implementation receives a packet that is well formed. That packet contains an Attribute allegedly of data type "address" but that has Length not equal to four. In that situation, the packet is well formed, but the Attribute is not. Therefore, it is an "invalid attribute".

A similar analysis can be performed when an attribute carries TLVs. The encapsulating attribute may be well formed, but the TLV may be an "invalid attribute". The existence of an "invalid attribute" in a packet or attribute MUST NOT result in the implementation discarding the entire packet or treating the packet as a negative acknowledgment. Instead, only the "invalid attribute" is treated specially.

When an implementation receives an "invalid attribute", it SHOULD be silently discarded, except when the implementation is acting as a proxy (see Section 5.2 for discussion of proxy servers). If it is

not discarded, it MUST NOT be handled in the same manner as a well-formed attribute. For example, receiving an Attribute of data type "address" containing either less than four octets or more than four octets of data means that the Attribute MUST NOT be treated as being of data type "address". The reason here is that if the Attribute does not carry an IPv4 address, the receiver has no idea what format the data is in, and it is therefore not an IPv4 address.

For Attributes of type "Long Extended Type", an Attribute is considered to be an "invalid attribute" when it does not match the criteria set out in Section 2.2, above.

For Attributes of type "TLV", an Attribute is considered to be an "invalid attribute" when the TLV-Length field allows the encapsulating Attribute to be parsed but the TLV-Value field does not match the criteria for that TLV. Implementations SHOULD NOT treat the "invalid attribute" property as being transitive. That is, the Attribute encapsulating the "invalid attribute" SHOULD NOT be treated as an "invalid attribute". That encapsulating Attribute might contain multiple TLVs, only one of which is an "invalid attribute".

However, a TLV definition may require particular sub-TLVs to be present and/or to have specific values. If a sub-TLV is missing or contains incorrect value(s), or if it is an "invalid attribute", then the encapsulating TLV SHOULD be treated as an "invalid attribute". This requirement ensures that strongly connected TLVs are either handled as a coherent whole or ignored entirely.

It is RECOMMENDED that Attributes with unknown Type, Extended-Type, TLV-Type, or EVS-Type are treated as "invalid attributes". This recommendation is compatible with the suggestion in [RFC2865] Section 5 that implementations "MAY ignore Attributes with an unknown Type".

3. Attribute Definitions

We define four (4) attributes of "Extended Type", which are allocated from the "Reserved" Attribute Type codes of 241, 242, 243, and 244. We also define two (2) attributes of "Long Extended Type", which are allocated from the "Reserved" Attribute Type codes of 245 and 246.

Type	Name
241	Extended-Type-1
242	Extended-Type-2
243	Extended-Type-3
244	Extended-Type-4
245	Long-Extended-Type-1
246	Long-Extended-Type-2

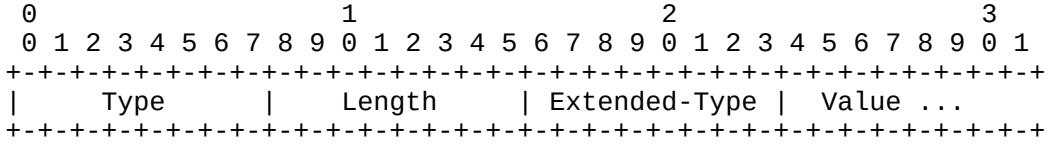
The rest of this section gives detailed definitions for each Attribute based on the above summary.

3.1. Extended-Type-1

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type space of 241.{1-255}.

A summary of the Extended-Type-1 Attribute format is shown below. The fields are transmitted from left to right.



Type

241 for Extended-Type-1.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 241.{1-255} RADIUS Attribute Type space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The "Value" field is one or more octets.

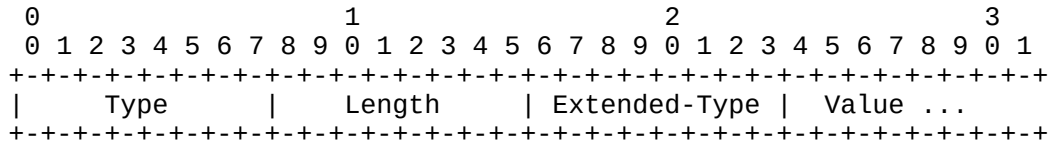
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type" to determine the interpretation of the "Value" field.

3.2. Extended-Type-2

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type space of 242.{1-255}.

A summary of the Extended-Type-2 Attribute format is shown below. The fields are transmitted from left to right.



Type

242 for Extended-Type-2.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 242.{1-255} RADIUS Attribute Type space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The "Value" field is one or more octets.

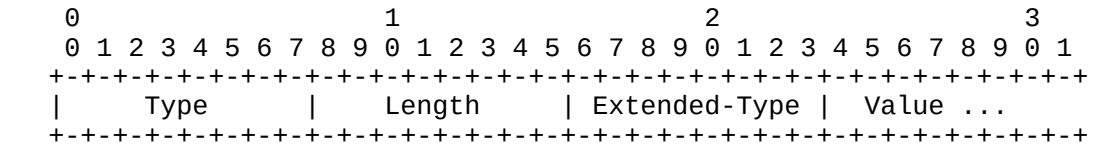
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type" to determine the interpretation of the "Value" field.

3.3. Extended-Type-3

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type space of 243.{1-255}.

A summary of the Extended-Type-3 Attribute format is shown below. The fields are transmitted from left to right.



Type

243 for Extended-Type-3.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 243.{1-255} RADIUS Attribute Type space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The "Value" field is one or more octets.

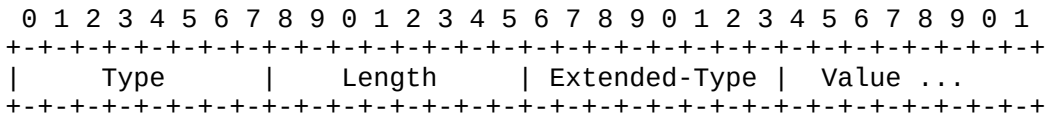
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type" to determine the interpretation of the "Value" field.

3.4. Extended-Type-4

Description

This attribute encapsulates attributes of the "Extended Type" format, in the RADIUS Attribute Type space of 244.{1-255}.

A summary of the Extended-Type-4 Attribute format is shown below. The fields are transmitted from left to right.



Type

244 for Extended-Type-4.

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 244.{1-255} RADIUS Attribute Type space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

Value

The "Value" field is one or more octets.

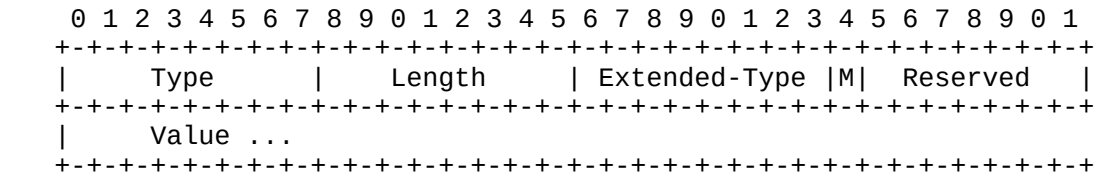
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type" to determine the interpretation of the Value Field.

3.5. Long-Extended-Type-1

Description

This attribute encapsulates attributes of the "Long Extended Type" format, in the RADIUS Attribute Type space of 245.{1-255}.

A summary of the Long-Extended-Type-1 Attribute format is shown below. The fields are transmitted from left to right.



Type

245 for Long-Extended-Type-1

Length

>= 5

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 245.{1-255} RADIUS Attribute Type space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

M (More)

The More field is one (1) bit in length and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Value

The "Value" field is one or more octets.

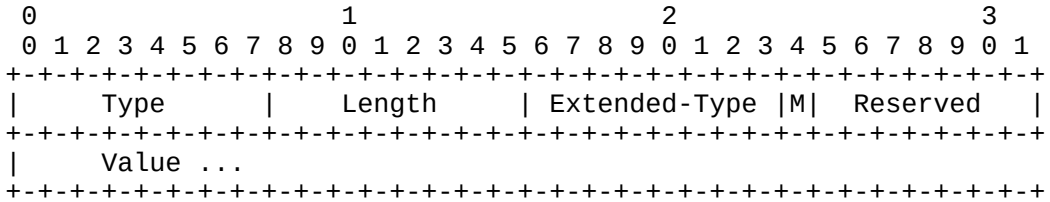
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type" to determine the interpretation of the "Value" field.

3.6. Long-Extended-Type-2

Description

This attribute encapsulates attributes of the "Long Extended Type" format, in the RADIUS Attribute Type space of 246.{1-255}.

A summary of the Long-Extended-Type-2 Attribute format is shown below. The fields are transmitted from left to right.



Type

246 for Long-Extended-Type-2

Length

>= 5

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified in the 246.{1-255} RADIUS Attribute Type space, according to the policies and rules described in Section 10. Further definition of this field is given in Section 2.1, above.

M (More)

The More field is one (1) bit in length and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Value

The "Value" field is one or more octets.

Implementations supporting this specification MUST use the identifier of "Type.Extended-Type" to determine the interpretation of the "Value" field.

4. Vendor-Specific Attributes

We define six new attributes that can carry vendor-specific information. We define four (4) attributes of the "Extended Type" format, with Type codes (241.26, 242.26, 243.26, 244.26), using the "evs" data type. We also define two (2) attributes using "Long Extended Type" format, with Type codes (245.26, 246.26), which are of the "evs" data type.

Type.Extended-Type	Name
241.26	Extended-Vendor-Specific-1
242.26	Extended-Vendor-Specific-2
243.26	Extended-Vendor-Specific-3
244.26	Extended-Vendor-Specific-4
245.26	Extended-Vendor-Specific-5
246.26	Extended-Vendor-Specific-6

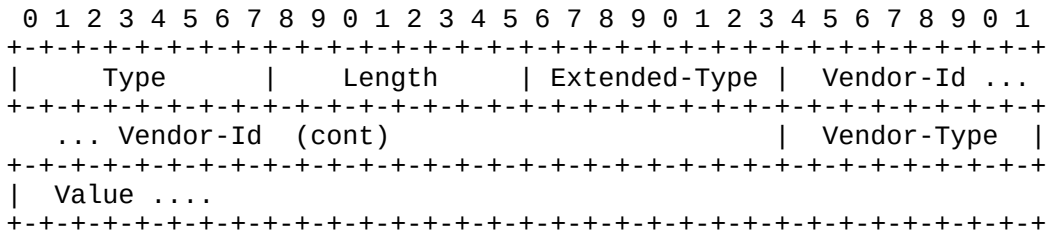
The rest of this section gives detailed definitions for each Attribute based on the above summary.

4.1. Extended-Vendor-Specific-1

Description

This attribute defines a RADIUS Type Code of 241.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-1 Attribute format is shown below. The fields are transmitted from left to right.



Type.Extended-Type

241.26 for Extended-Vendor-Specific-1

Length

>= 9

Vendor-Id

The 4 octets of the Vendor-Id field are the Network Management Private Enterprise Code [PEN] of the vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the vendor.

Value

The "Value" field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

The length of the "Value" field is eight (8) less than the value of the Length field.

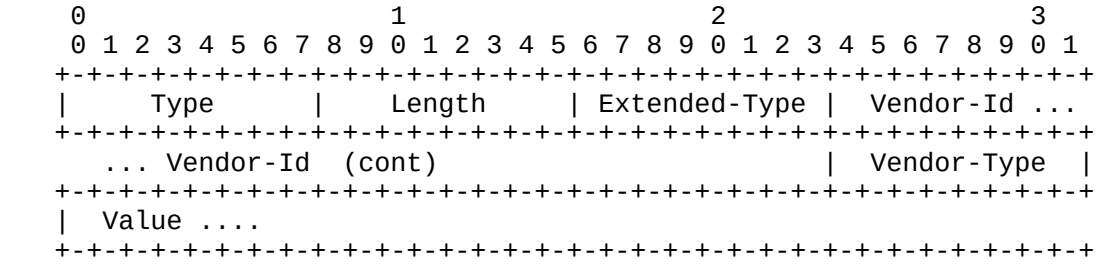
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the "Value" field.

4.2. Extended-Vendor-Specific-2

Description

This attribute defines a RADIUS Type Code of 242.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-2 Attribute format is shown below. The fields are transmitted from left to right.



Type.Extended-Type

242.26 for Extended-Vendor-Specific-2

Length

>= 9

Vendor-Id

The 4 octets of the Vendor-Id field are the Network Management Private Enterprise Code [PEN] of the vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the vendor.

Value

The "Value" field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

The length of the "Value" field is eight (8) less than the value of the Length field.

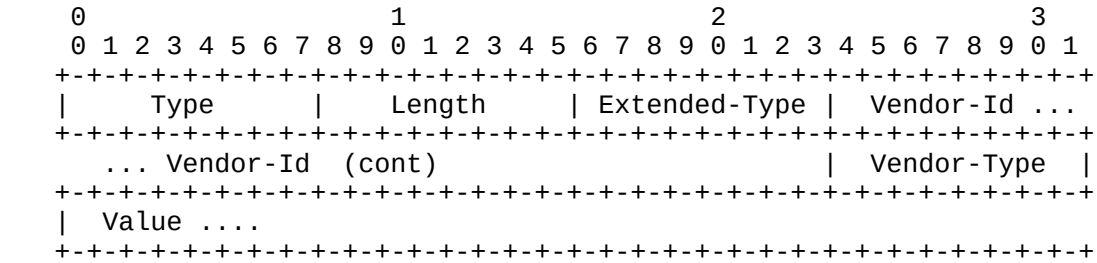
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the "Value" field.

4.3. Extended-Vendor-Specific-3

Description

This attribute defines a RADIUS Type Code of 243.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-3 Attribute format is shown below. The fields are transmitted from left to right.



Type.Extended-Type

243.26 for Extended-Vendor-Specific-3

Length

>= 9

Vendor-Id

The 4 octets of the Vendor-Id field are the Network Management Private Enterprise Code [PEN] of the vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the vendor.

Value

The "Value" field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

The length of the "Value" field is eight (8) less than the value of the Length field.

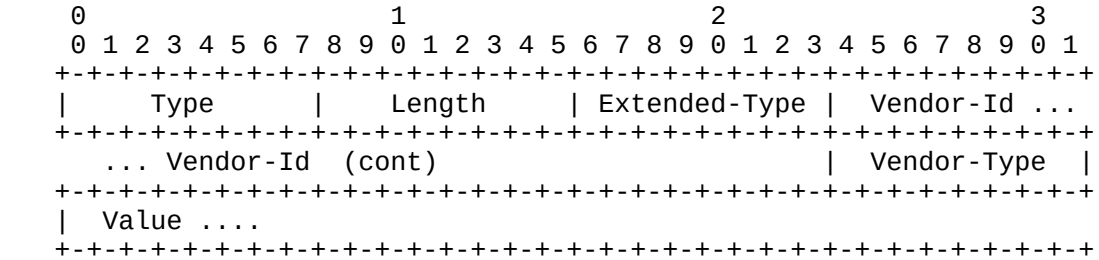
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the "Value" field.

4.4. Extended-Vendor-Specific-4

Description

This attribute defines a RADIUS Type Code of 244.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-4 Attribute format is shown below. The fields are transmitted from left to right.



Type.Extended-Type

244.26 for Extended-Vendor-Specific-4

Length

>= 9

Vendor-Id

The 4 octets of the Vendor-Id field are the Network Management Private Enterprise Code [PEN] of the vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the vendor.

Value

The "Value" field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

The length of the "Value" field is eight (8) less than the value of the Length field.

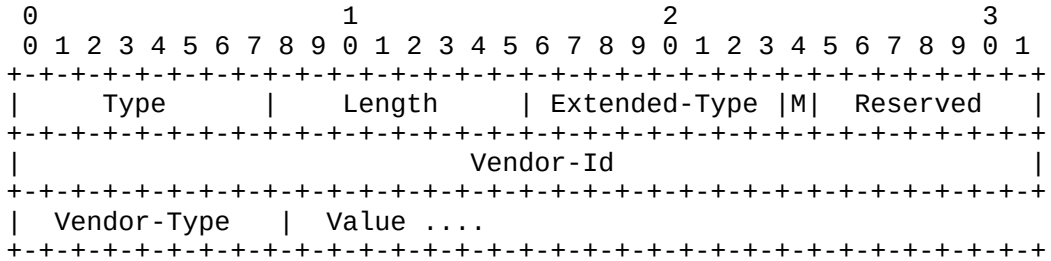
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the "Value" field.

4.5. Extended-Vendor-Specific-5

Description

This attribute defines a RADIUS Type Code of 245.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-5 Attribute format is shown below. The fields are transmitted from left to right.



Type.Extended-Type

245.26 for Extended-Vendor-Specific-5

Length

>= 10 (first fragment)
>= 5 (subsequent fragments)

When a VSA is fragmented across multiple Attributes, only the first Attribute contains the Vendor-Id and Vendor-Type fields. Subsequent Attributes contain fragments of the "Value" field only.

M (More)

The More field is one (1) bit in length and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Vendor-Id

The 4 octets of the Vendor-Id field are the Network Management Private Enterprise Code [PEN] of the vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the vendor.

Value

The "Value" field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

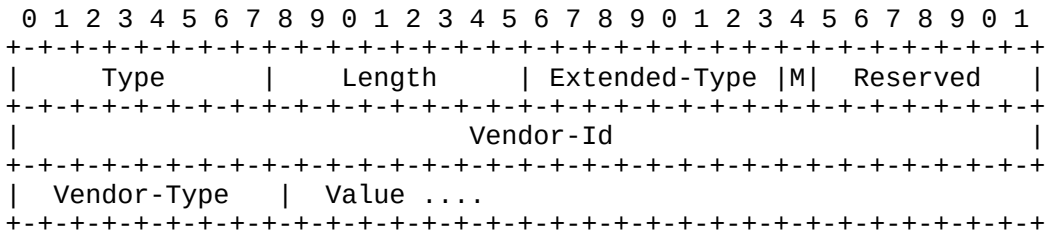
Implementations supporting this specification MUST use the identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the "Value" field.

4.6. Extended-Vendor-Specific-6

Description

This attribute defines a RADIUS Type Code of 246.26, using the "evs" data type.

A summary of the Extended-Vendor-Specific-6 Attribute format is shown below. The fields are transmitted from left to right.



Type.Extended-Type

246.26 for Extended-Vendor-Specific-6

Length

>= 10 (first fragment)
>= 5 (subsequent fragments)

When a VSA is fragmented across multiple Attributes, only the first Attribute contains the Vendor-Id and Vendor-Type fields. Subsequent Attributes contain fragments of the "Value" field only.

M (More)

The More field is one (1) bit in length and indicates whether or not the current attribute contains "more" than 251 octets of data. Further definition of this field is given in Section 2.2, above.

Reserved

This field is 7 bits long and is reserved for future use. Implementations MUST set it to zero (0) when encoding an attribute for sending in a packet. The contents SHOULD be ignored on reception.

Vendor-Id

The 4 octets of the Vendor-Id field are the Network Management Private Enterprise Code [PEN] of the vendor in network byte order.

Vendor-Type

The Vendor-Type field is one octet. Values are assigned at the sole discretion of the vendor.

Value

The "Value" field is one or more octets. The actual format of the information is site or application specific, and a robust implementation SHOULD support the field as undistinguished octets.

The codification of the range of allowed usage of this field is outside the scope of this specification.

Implementations supporting this specification MUST use the identifier of "Type.Extended-Type.Vendor-Id.Vendor-Type" to determine the interpretation of the "Value" field.

5. Compatibility with Traditional RADIUS

There are a number of potential compatibility issues with traditional RADIUS, as defined in [RFC6158] and earlier. This section describes them.

5.1. Attribute Allocation

Some vendors have used Attribute Type codes from the "Reserved" space as part of vendor-defined dictionaries. This practice is considered antisocial behavior, as noted in [RFC6158]. These vendor definitions conflict with the Attributes in the RADIUS Attribute Type space. The conflicting definitions may make it difficult for implementations to support both those Vendor Attributes, and the new Extended Attribute formats.

We RECOMMEND that RADIUS client and server implementations delete all references to these improperly defined attributes. Failing that, we RECOMMEND that RADIUS server implementations have a per-client configurable flag that indicates which type of attributes are being sent from the client. If the flag is set to "Non-Standard Attributes", the conflicting attributes can be interpreted as being improperly defined Vendor-Specific Attributes. If the flag is set to

"IETF Attributes", the Attributes MUST be interpreted as being of the Extended Attributes format. The default SHOULD be to interpret the Attributes as being of the Extended Attributes format.

Other methods of determining how to decode the Attributes into a "correct" form are NOT RECOMMENDED. Those methods are likely to be fragile and prone to error.

We RECOMMEND that RADIUS server implementations reuse the above flag to determine which types of attributes to send in a reply message. If the request is expected to contain the improperly defined attributes, the reply SHOULD NOT contain Extended Attributes. If the request is expected to contain Extended Attributes, the reply MUST NOT contain the improper Attributes.

RADIUS clients will have fewer issues than servers. Clients MUST NOT send improperly defined Attributes in a request. For replies, clients MUST interpret attributes as being of the Extended Attributes format, instead of the improper definitions. These requirements impose no change in the RADIUS specifications, as such usage by vendors has always been in conflict with the standard requirements and the standards process.

Existing clients that send these improperly defined attributes usually have a configuration setting that can disable this behavior. We RECOMMEND that vendors ship products with the default set to "disabled". We RECOMMEND that administrators set this flag to "disabled" on all equipment that they manage.

5.2. Proxy Servers

RADIUS proxy servers will need to forward Attributes having the new format, even if they do not implement support for the encoding and decoding of those attributes. We remind implementers of the following text in [RFC2865] Section 2.3:

> The forwarding server MUST NOT change the order of any attributes of the same type, including Proxy-State.

This requirement solves some of the issues related to proxying of the new format, but not all. The reason is that proxy servers are permitted to examine the contents of the packets that they forward. Many proxy implementations not only examine the Attributes, but they refuse to forward attributes that they do not understand (i.e., attributes for which they have no local dictionary definitions).

This practice is NOT RECOMMENDED. Proxy servers SHOULD forward attributes, even attributes that they do not understand or that are not in a local dictionary. When forwarded, these attributes SHOULD be sent verbatim, with no modifications or changes. This requirement includes "invalid attributes", as there may be some other system in the network that understands them.

The only exception to this recommendation is when local site policy dictates that filtering of attributes has to occur. For example, a filter at a visited network may require removal of certain authorization rules that apply to the home network but not to the visited network. This filtering can sometimes be done even when the contents of the Attributes are unknown, such as when all Vendor-Specific Attributes are designated for removal.

As seen during testing performed in 2010 via the EDUcation ROAMing (EDUROAM) service (A. DeKok, unpublished data), many proxies do not follow these practices for unknown Attributes. Some proxies filter out unknown attributes or attributes that have unexpected lengths (24%, 17/70), some truncate the Attributes to the "expected" length (11%, 8/70), some discard the request entirely (1%, 1/70), and the rest (63%, 44/70) follow the recommended practice of passing the Attributes verbatim. It will be difficult to widely use the Extended Attributes format until all non-conformant proxies are fixed. We therefore RECOMMEND that all proxies that do not support the Extended Attributes (241 through 246) define them as being of data type "string" and delete all other local definitions for those attributes.

This last change should enable wider usage of the Extended Attributes format.

6. Guidelines

This specification proposes a number of changes to RADIUS and therefore requires a set of guidelines, as has been done in [RFC6158]. These guidelines include suggestions related to design, interaction with IANA, usage, and implementation of attributes using the new formats.

6.1. Updates to RFC 6158

This specification updates [RFC6158] by adding the data types "evs", "tlv", and "integer64"; defining them to be "basic" data types; and permitting their use subject to the restrictions outlined below.

The recommendations for the use of the new data types and Attribute formats are given below.

6.2. Guidelines for Simple Data Types

[RFC6158] Section A.2.1 says in part:

* Unsigned integers of size other than 32 bits. SHOULD be replaced by an unsigned integer of 32 bits. There is insufficient justification to define a new size of integer.

We update that specification to permit unsigned integers of 64 bits, for the reasons defined above in Section 2.5. The updated text is as follows:

* Unsigned integers of size other than 32 or 64 bits. SHOULD be replaced by an unsigned integer of 32 or 64 bits. There is insufficient justification to define a new size of integer.

That section later continues with the following list item:

* Nested attribute-value pairs (AVPs). Attributes should be defined in a flat typespace.

We update that specification to permit nested TLVs, as defined in this document:

* Nested attribute-value pairs (AVPs) using the extended Attribute format MAY be used. All other nested AVP or TLV formats MUST NOT be used.

The [RFC6158] recommendations for "basic" data types apply to the three types listed above. All other recommendations given in [RFC6158] for "basic" data types remain unchanged.

6.3. Guidelines for Complex Data Types

[RFC6158] Section 2.1 says:

> Complex data types MAY be used in situations where they reduce complexity in non-RADIUS systems or where using the basic data types would be awkward (such as where grouping would be required in order to link related attributes).

Since the extended Attribute format allows for grouping of complex types via TLVs, the guidelines for complex data types need to be updated as follows:

> [RFC6158], Section 3.2.4, describes situations in which complex data types might be appropriate. They SHOULD NOT be used even in those situations, without careful consideration of the described

limitations. In all other cases not covered by the complex data type exceptions, complex data types MUST NOT be used. Instead, complex data types MUST be decomposed into TLVs.

The checklist in [RFC6158] Appendix A.2.2 is similarly updated to add a new requirement at the top of that section, as follows:

Does the Attribute

* define a complex type that can be represented via TLVs?

If so, this data type MUST be represented via TLVs.

Note that this requirement does not override [RFC6158] Appendix A.1, which permits the transport of complex types in certain situations.

All other recommendations given in [RFC6158] for "complex" data types remain unchanged.

6.4. Design Guidelines for the New Types

This section gives design guidelines for specifications defining attributes using the new format. The items listed below are not exhaustive. As experience is gained with the new formats, later specifications may define additional guidelines.

* The data type "evs" MUST NOT be used for standard RADIUS Attributes, or for TLVs, or for VSAs.

* The data type TLV SHOULD NOT be used for standard RADIUS attributes.

* [RFC2866] "tagged" attributes MUST NOT be defined in the Extended-Type space. The "tlv" data type should be used instead to group attributes.

* The "integer64" data type MAY be used in any RADIUS attribute. The use of 64-bit integers was not recommended in [RFC6158], but their utility is now evident.

* Any attribute that is allocated from the long extended space of data type "text", "string", or "tlv" can potentially carry more than 251 octets of data. Specifications defining such attributes SHOULD define a maximum length to guide implementations.

All other recommendations given in [RFC6158] for attribute design guidelines apply to attributes using the short extended space and long extended space.

6.5. TLV Guidelines

The following items give design guidelines for specifications using TLVs.

* When multiple Attributes are intended to be grouped or managed together, the use of TLVs to group related attributes is RECOMMENDED.

* More than 4 layers (depth) of TLV nesting is NOT RECOMMENDED.

* Interpretation of an attribute depends only on its type definition (e.g., Type.Extended-Type.TLV-Type) and not on its encoding or location in the RADIUS packet.

* Where a group of TLVs is strictly defined, and not expected to change, and totals less than 247 octets of data, the specifications SHOULD request allocation from the short extended space.

* Where a group of TLVs is loosely defined or is expected to change, the specifications SHOULD request allocation from the long extended space.

All other recommendations given in [RFC6158] for attribute design guidelines apply to attributes using the TLV format.

6.6. Allocation Request Guidelines

The following items give guidelines for allocation requests made in a RADIUS specification.

* Discretion is recommended when requesting allocation of attributes. The new space is much larger than the old one, but it is not infinite.

* Specifications that allocate many attributes MUST NOT request that allocation be made from the standard space. That space is under allocation pressure, and the extended space is more suitable for large allocations. As a guideline, we suggest that one specification allocating twenty percent (20%) or more of the standard space would meet the above criteria.

* Specifications that allocate many related attributes SHOULD define one or more TLVs to contain related attributes.

* Specifications SHOULD request allocation from a specific space. The IANA considerations given in Section 10, below, give instructions to IANA, but authors should assist IANA where possible.

* Specifications of an attribute that encodes 252 octets or less of data MAY request allocation from the short extended space.

* Specifications of an attribute that always encode less than 253 octets of data MUST NOT request allocation from the long extended space. The standard space or the short extended space MUST be used instead.

* Specifications of an attribute that encodes 253 octets or more of data MUST request allocation from the long extended space.

* When the extended space is nearing exhaustion, a new specification will have to be written that requests allocation of one or more RADIUS attributes from the "Reserved" portion of the standard space, values 247-255, using an appropriate format ("Short Extended Type", or "Long Extended Type").

An allocation request made in a specification SHOULD use one of the following formats when allocating an attribute type code:

* TBDn - request allocation of an attribute from the standard space. The value "n" should be 1 or more, to track individual attributes that are to be allocated.

* SHORT-TBDn - request allocation of an attribute from the short extended space. The value "n" should be 1 or more, to track individual attributes that are to be allocated.

* LONG-TBDn - request allocation of an attribute from the long extended space. The value "n" should be 1 or more, to track individual attributes that are to be allocated.

These guidelines should help specification authors and IANA communicate effectively and clearly.

6.7. Allocation Request Guidelines for TLVs

Specifications may allocate a new attribute of type TLV and at the same time allocate sub-Attributes within that TLV. These specifications SHOULD request allocation of specific values for the sub-TLV. The "dotted number" notation MUST be used.

For example, a specification may request allocation of a TLV as SHORT-TBD1. Within that attribute, it could request allocation of three sub-TLVs, as SHORT-TBD1.1, SHORT-TBD1.2, and SHORT-TBD1.3.

Specifications may request allocation of additional sub-TLVs within an existing attribute of type TLV. Those specifications SHOULD use the "TBDn" format for every entry in the "dotted number" notation.

For example, a specification may request allocation within an existing TLV, with "dotted number" notation MM.NN. Within that attribute, the specification could request allocation of three sub-TLVs, as MM.NN.TBD1, MM.NN.TBD2, and MM.NN.TBD3.

6.8. Implementation Guidelines

* RADIUS client implementations SHOULD support this specification in order to permit the easy deployment of specifications using the changes defined herein.

* RADIUS server implementations SHOULD support this specification in order to permit the easy deployment of specifications using the changes defined herein.

* RADIUS proxy servers MUST follow the specifications in Section 5.2.

6.9. Vendor Guidelines

* Vendors SHOULD use the existing Vendor-Specific Attribute Type space in preference to the new Extended-Vendor-Specific Attributes, as this specification may take time to become widely deployed.

* Vendors SHOULD implement this specification. The changes to RADIUS are relatively small and are likely to quickly be used in new specifications.

7. Rationale for This Design

The path to extending the RADIUS protocol has been long and arduous. A number of proposals have been made and discarded by the RADEXT working group. These proposals have been judged to be either too bulky, too complex, too simple, or unworkable in practice. We do not otherwise explain here why earlier proposals did not obtain working group consensus.

The changes outlined here have the benefit of being simple, as the "Extended Type" format requires only a one-octet change to the Attribute format. The downside is that the "Long Extended Type" format is awkward, and the 7 Reserved bits will likely never be used for anything.

7.1. Attribute Audit

An audit of almost five thousand publicly available attributes [ATTR] (2010) shows the statistics summarized below. The Attributes include over 100 Vendor dictionaries, along with the IANA-assigned attributes:

Count	Data Type
2257	integer
1762	text
273	IPv4 Address
225	string
96	other data types
35	IPv6 Address
18	date
10	integer64
4	Interface Id
3	IPv6 Prefix
4683	Total

The entries in the "Data Type" column are data types recommended by [RFC6158], along with "integer64". The "other data types" row encompasses all other data types, including complex data types and data types transporting opaque data.

We see that over half of the Attributes encode less than 16 octets of data. It is therefore important to have an extension mechanism that adds as little as possible to the size of these attributes. Another result is that the overwhelming majority of attributes use simple data types.

Of the Attributes defined above, 177 were declared as being inside of a TLV. This is approximately 4% of the total. We did not investigate whether additional attributes were defined in a flat namespace but could have been defined as being inside of a TLV. We expect that the number could be as high as 10% of attributes.

Manual inspection of the dictionaries shows that approximately 20 (or 0.5%) attributes have the ability to transport more than 253 octets of data. These attributes are divided between VSAs and a small number of standard Attributes such as EAP-Message.

The results of this audit and analysis are reflected in the design of the extended attributes. The extended format has minimal overhead, permits TLVs, and has support for "long" attributes.

8. Diameter Considerations

The Attribute formats defined in this specification need to be transported in Diameter. While Diameter supports attributes longer than 253 octets and grouped attributes, we do not use that functionality here. Instead, we define the simplest possible encapsulation method.

The new formats MUST be treated the same as traditional RADIUS attributes when converting from RADIUS to Diameter, or vice versa. That is, the new attribute space is not converted to any "extended" Diameter attribute space. Fragmented attributes are not converted to a single long Diameter attribute. The new EVS data types are not converted to Diameter attributes with the "V" bit set.

In short, this document mandates no changes for existing RADIUS-to-Diameter or Diameter-to-RADIUS gateways.

9. Examples

A few examples are presented here in order to illustrate the encoding of the new Attribute formats. These examples are not intended to be exhaustive, as many others are possible. For simplicity, we do not show complete packets, but only attributes.

The examples are given using a domain-specific language implemented by the program given in Appendix A of this document. The language is line oriented and composed of a sequence of lines matching the ABNF grammar ([RFC5234]) given below:

```
Identifier = 1*DIGIT *( "." 1*DIGIT )

HEXCHAR = HEXDIG HEXDIG

STRING = DQUOTE 1*CHAR DQUOTE

TLV = "{" SP 1*DIGIT SP DATA SP "}"

DATA = (HEXCHAR *(SP HEXCHAR)) / (TLV *(SP TLV)) / STRING

LINE = Identifier SP DATA
```

The program has additional restrictions on its input that are not reflected in the above grammar. For example, the portions of the identifier that refer to Type and Extended-Type are limited to values between 1 and 255. We trust that the source code in Appendix A is clear and that these restrictions do not negatively affect the comprehensibility of the examples.

The program reads the input text and interprets it as a set of instructions to create RADIUS attributes. It then prints the hex encoding of those attributes. It implements the minimum set of functionality that achieves that goal. This minimalism means that it does not use attribute dictionaries; it does not implement support for RADIUS data types; it can be used to encode attributes with invalid data fields; and there is no requirement for consistency from one example to the next. For example, it can be used to encode a User-Name attribute that contains non-UTF8 data or a Framed-IP-Address that contains 253 octets of ASCII data. As a result, it MUST NOT be used to create RADIUS attributes for transport in a RADIUS message.

However, the program correctly encodes the RADIUS attribute fields of "Type", "Length", "Extended-Type", "More", "Reserved", "Vendor-Id", "Vendor-Type", and "Vendor-Length". It encodes RADIUS attribute data types "evs" and "tlv". It can therefore be used to encode example attributes from inputs that are human readable.

We do not give examples of "invalid attributes". We also note that the examples show format, rather than consistent meaning. A particular Attribute Type code may be used to demonstrate two different formats. In real specifications, attributes have a static definitions based on their type code.

The examples given below are strictly for demonstration purposes only and do not provide a standard of any kind.

9.1. Extended Type

The following is a series of examples of the "Extended Type" format.

Attribute encapsulating textual data:

```
241.1 "bob"
  -> f1 06 01 62 6f 62
```

Attribute encapsulating a TLV with TLV-Type of one (1):

```
241.2 { 1 23 45 }
  -> f1 07 02 01 04 23 45
```

Attribute encapsulating two TLVs, one after the other:

```
241.2 { 1 23 45 } { 2 67 89 }
  -> f1 0b 02 01 04 23 45 02 04 67 89
```

Attribute encapsulating two TLVs, where the second TLV is itself encapsulating a TLV:

```
241.2 { 1 23 45 } { 3 { 1 ab cd } }
  -> f1 0d 02 01 04 23 45 03 06 01 04 ab cd
```

Attribute encapsulating two TLVs, where the second TLV is itself encapsulating two TLVs:

```
241.2 { 1 23 45 } { 3 { 1 ab cd } { 2 "foo" } }
  -> f1 12 02 01 04 23 45 03 0b 01 04 ab cd 02 05 66 6f 6f
```

Attribute encapsulating a TLV, which in turn encapsulates a TLV, to a depth of 5 nestings:

```
241.1 { 1 { 2 { 3 { 4 { 5 cd ef } } } } }
  -> f1 0f 01 01 0c 02 0a 03 08 04 06 05 04 cd ef
```

Attribute encapsulating an Extended-Vendor-Specific Attribute, with Vendor-Id of 1 and Vendor-Type of 4, which in turn encapsulates textual data:

```
241.26.1.4 "test"
  -> f1 0c 1a 00 00 00 01 04 74 65 73 74
```

Attribute encapsulating an Extended-Vendor-Specific Attribute, with Vendor-Id of 1 and Vendor-Type of 5, which in turn encapsulates a TLV with TLV-Type of 3, which encapsulates textual data:

```
241.26.1.5 { 3 "test" }
  -> f1 0e 1a 00 00 00 01 05 03 06 74 65 73 74
```

9.2. Long Extended Type

The following is a series of examples of the "Long Extended Type" format.

Attribute encapsulating textual data:

```
245.1 "bob"
  -> f5 07 01 00 62 6f 62
```

Attribute encapsulating a TLV with TLV-Type of one (1):

```
245.2 { 1 23 45 }
  -> f5 08 02 00 01 04 23 45
```

Attribute encapsulating two TLVs, one after the other:

```
245.2 { 1 23 45 } { 2 67 89 }
  -> f5 0c 02 00 01 04 23 45 02 04 67 89
```

Attribute encapsulating two TLVs, where the second TLV is itself encapsulating a TLV:

```
245.2 { 1 23 45 } { 3 { 1 ab cd } }
  -> f5 0e 02 00 01 04 23 45 03 06 01 04 ab cd
```

Attribute encapsulating two TLVs, where the second TLV is itself encapsulating two TLVs:

```
245.2 { 1 23 45 } { 3 { 1 ab cd } { 2 "foo" } }
  -> f5 13 02 00 01 04 23 45 03 0b 01 04 ab cd 02 05 66 6f 6f
```

Attribute encapsulating a TLV, which in turn encapsulates a TLV, to a depth of 5 nestings:

```
245.1 { 1 { 2 { 3 { 4 { 5 cd ef } } } } }
  -> f5 10 01 00 01 0c 02 0a 03 08 04 06 05 04 cd ef
```

Attribute encapsulating an Extended-Vendor-Specific Attribute, with Vendor-Id of 1 and Vendor-Type of 4, which in turn encapsulates textual data:

```
245.26.1.4 "test"
  -> f5 0d 1a 00 00 00 00 01 04 74 65 73 74
```

Attribute encapsulating an Extended-Vendor-Specific Attribute, with Vendor-Id of 1 and Vendor-Type of 5, which in turn encapsulates a TLV with TLV-Type of 3, which encapsulates textual data:

```
245.26.1.5 { 3 "test" }
  -> f5 0f 1a 00 00 00 00 01 05 03 06 74 65 73 74
```

Attribute encapsulating more than 251 octets of data. The "Data" portions are indented for readability:

```
245.4 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
      bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
      bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
      bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
      bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbccccccccccccccccccccc
      ccccccccccc"
  -> f5 ff 04 80 aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa ab bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb f5 13 04 00 cc
     cc cc cc cc cc cc cc cc cc cc cc cc cc cc
```

Below is an example of an attribute encapsulating an Extended-Vendor-Specific Attribute, with Vendor-Id of 1 and Vendor-Type of 6, which in turn encapsulates more than 251 octets of data.

As the VSA encapsulates more than 251 octets of data, it is split into two RADIUS attributes. The first attribute has the More field set, and it carries the Vendor-Id and Vendor-Type. The second attribute has the More field clear and carries the rest of the data portion of the VSA. Note that the second attribute does not include the Vendor-Id ad Vendor-Type fields.

The "Data" portions are indented for readability:

```
245.26.1.6  "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
      bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
      bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
      bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
      bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbccccccccccccccc
      ccccccccccccccccc"
  -> f5 ff 1a 80 00 00 00 01 06 aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
     aa aa aa aa aa aa aa aa aa aa aa aa aa aa ab bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
     bb bb bb bb bb bb bb bb bb bb bb bb bb bb f5 18 1a 00 bb
     bb bb bb bb cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
```

10. IANA Considerations

This document updates [RFC3575] in that it adds new IANA considerations for RADIUS attributes. These considerations modify and extend the IANA considerations for RADIUS, rather than replacing them.

The IANA considerations of this document are limited to the "RADIUS Attribute Types" registry. Some Attribute Type values that were previously marked "Reserved" are now allocated, and the registry is extended from a simple 8-bit array to a tree-like structure, up to a maximum depth of 125 nodes. Detailed instructions are given below.

10.1. Attribute Allocations

IANA has moved the following Attribute Type values from "Reserved" to "Allocated" with the corresponding names:

* 241 Extended-Type-1
* 242 Extended-Type-2
* 243 Extended-Type-3
* 244 Extended-Type-4
* 245 Long-Extended-Type-1
* 246 Long-Extended-Type-2

These values serve as an encapsulation layer for the new RADIUS Attribute Type tree.

10.2. RADIUS Attribute Type Tree

Each of the Attribute Type values allocated above extends the "RADIUS Attribute Types" to an N-ary tree, via a "dotted number" notation. Allocation of an Attribute Type value "TYPE" using the new "Extended Type" format results in allocation of 255 new Attribute Type values of format "TYPE.1" through "TYPE.255". Value twenty-six (26) is assigned as "Extended-Vendor-Specific-*". Values "TYPE.241" through "TYPE.255" are marked "Reserved". All other values are "Unassigned".

The initial set of Attribute Type values and names assigned by this document is given below.

* 241 Extended-Attribute-1
* 241.{1-25} Unassigned
* 241.26 Extended-Vendor-Specific-1
* 241.{27-240} Unassigned
* 241.{241-255} Reserved
* 242 Extended-Attribute-2
* 242.{1-25} Unassigned
* 242.26 Extended-Vendor-Specific-2
* 242.{27-240} Unassigned
* 242.{241-255} Reserved
* 243 Extended-Attribute-3
* 243.{1-25} Unassigned
* 243.26 Extended-Vendor-Specific-3
* 243.{27-240} Unassigned
* 243.{241-255} Reserved
* 244 Extended-Attribute-4
* 244.{1-25} Unassigned
* 244.26 Extended-Vendor-Specific-4
* 244.{27-240} Unassigned
* 244.{241-255} Reserved
* 245 Extended-Attribute-5
* 245.{1-25} Unassigned
* 245.26 Extended-Vendor-Specific-5
* 245.{27-240} Unassigned
* 245.{241-255} Reserved
* 246 Extended-Attribute-6
* 246.{1-25} Unassigned
* 246.26 Extended-Vendor-Specific-6
* 246.{27-240} Unassigned
* 246.{241-255} Reserved

As per [RFC5226], the values marked "Unassigned" above are available for assignment by IANA in future RADIUS specifications. The values marked "Reserved" are reserved for future use.

The Extended-Vendor-Specific spaces (TYPE.26) are for Private Use, and allocations are not managed by IANA.

Allocation of Reserved entries in the extended space requires Standards Action.

All other allocations in the extended space require IETF Review.

10.3. Allocation Instructions

This section defines what actions IANA needs to take when allocating new attributes. Different actions are required when allocating attributes from the standard space, attributes of the "Extended Type" format, attributes of the "Long Extended Type" format, preferential allocations, attributes of data type TLV, attributes within a TLV, and attributes of other data types.

10.3.1. Requested Allocation from the Standard Space

Specifications can request allocation of an Attribute from within the standard space (e.g., Attribute Type Codes 1 through 255), subject to the considerations of [RFC3575] and this document.

10.3.2. Requested Allocation from the Short Extended Space

Specifications can request allocation of an Attribute that requires the format "Extended Type", by specifying the short extended space. In that case, IANA should assign the lowest Unassigned number from the Attribute Type space with the relevant format.

10.3.3. Requested Allocation from the Long Extended Space

Specifications can request allocation of an Attribute that requires the format "Long Extended Type", by specifying the extended space (long). In that case, IANA should assign the lowest Unassigned number from the Attribute Type space with the relevant format.

10.3.4. Allocation Preferences

Specifications that make no request for allocation from a specific type space should have Attributes allocated using the following criteria:

* When the standard space has no more Unassigned attributes, all allocations should be performed from the extended space.

* Specifications that allocate a small number of attributes (i.e., less than ten) should have all allocations made from the standard space.

* Specifications that would allocate more than twenty percent of the remaining standard space attributes should have all allocations made from the extended space.

* Specifications that request allocation of an attribute of data type TLV should have that attribute allocated from the extended space.

* Specifications that request allocation of an attribute that can transport 253 or more octets of data should have that attribute allocated from within the long extended space. We note that Section 6.5 above makes recommendations related to this allocation.

There is otherwise no requirement that all attributes within a specification be allocated from one type space or another. Specifications can simultaneously allocate attributes from both the standard space and the extended space.

10.3.5. Extending the Type Space via the TLV Data Type

When specifications request allocation of an attribute of data type TLV, that allocation extends the Attribute Type tree by one more level. Allocation of an Attribute Type value "TYPE.TLV", with data type TLV, results in allocation of 255 new Attribute Type values, of format "TYPE.TLV.1" through "TYPE.TLV.255". Values 254-255 are marked "Reserved". All other values are "Unassigned". Value 26 has no special meaning.

For example, if a new attribute "Example-TLV" of data type TLV is assigned the identifier "245.1", then the extended tree will be allocated as below:

* 245.1 Example-TLV
* 245.1.{1-253} Unassigned
* 245.1.{254-255} Reserved

Note that this example does not define an "Example-TLV" attribute.

The Attribute Type tree can be extended multiple levels in one specification when the specification requests allocation of nested TLVs, as discussed below.

10.3.6. Allocation within a TLV

Specifications can request allocation of Attribute Type values within an Attribute of data type TLV. The encapsulating TLV can be allocated in the same specification, or it can have been previously allocated.

Specifications need to request allocation within a specific Attribute Type value (e.g., "TYPE.TLV.*"). Allocations are performed from the smallest Unassigned value, proceeding to the largest Unassigned value.

Where the Attribute being allocated is of data type TLV, the Attribute Type tree is extended by one level, as given in the previous section. Allocations can then be made within that level.

10.3.7. Allocation of Other Data Types

Attribute Type value allocations are otherwise allocated from the smallest Unassigned value, proceeding to the largest Unassigned value, e.g., starting from 241.1, proceeding through 241.255, then to 242.1, through 242.255, etc.

11. Security Considerations

This document defines new formats for data carried inside of RADIUS but otherwise makes no changes to the security of the RADIUS protocol.

Attacks on cryptographic hashes are well known and are getting better with time, as discussed in [RFC4270]. The security of the RADIUS protocol is dependent on MD5 [RFC1321], which has security issues as discussed in [RFC6151]. It is not known if the issues described in [RFC6151] apply to RADIUS. For other issues, we incorporate by reference the security considerations of [RFC6158] Section 5.

As with any protocol change, code changes are required in order to implement the new features. These code changes have the potential to introduce new vulnerabilities in the software. Since the RADIUS server performs network authentication, it is an inviting target for attackers. We RECOMMEND that access to RADIUS servers be kept to a minimum.

12. References

12.1. Normative References

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

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

[RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.

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

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

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

[PEN] IANA, "PRIVATE ENTERPRISE NUMBERS", <http://www.iana.org/assignments/enterprise-numbers>.

12.2. Informative References

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC2868] Zorn, G., Leifer, D., Rubens, A., Shriver, J., Holdrege, M., and I. Goyret, "RADIUS Attributes for Tunnel Protocol Support", RFC 2868, June 2000.

[RFC4270] Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.

[ATTR] "alandekok/freeradius-server", available from GitHub, data retrieved September 2010, <http://github.com/alandekok/freeradius-server/tree/master/share/>.

13. Acknowledgments

This document is the result of long discussions in the IETF RADEXT working group. The authors would like to thank all of the participants who contributed various ideas over the years. Their feedback has been invaluable and has helped to make this specification better.

Appendix A. Extended Attribute Generator Program

This section contains "C" program source code that can be used for testing. It reads a line-oriented text file, parses it to create RADIUS formatted attributes, and prints the hex version of those attributes to standard output.

The input accepts grammar similar to that given in Section 9, with some modifications for usability. For example, blank lines are allowed, lines beginning with a '#' character are interpreted as comments, numbers (RADIUS Types, etc.) are checked for minimum/maximum values, and RADIUS attribute lengths are enforced.

The program is included here for demonstration purposes only, and does not define a standard of any kind.

```
------------------------------------------------------------
/*
 * Copyright (c) 2013 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 *
 * - Redistributions in binary form must reproduce the above
 *   copyright notice, this list of conditions and the following
 *   disclaimer in the documentation and/or other materials provided
 *   with the distribution.
 *
 * - Neither the name of Internet Society, IETF or IETF Trust, nor
 *   the names of specific contributors, may be used to endorse or
 *   promote products derived from this software without specific
 *   prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
 * CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
```

```
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *  Author:  Alan DeKok <aland@networkradius.com>
 */
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>

static int encode_tlv(char *buffer, uint8_t *output, size_t outlen);

static const char *hextab = "0123456789abcdef";

static int encode_data_string(char *buffer,
                      uint8_t *output, size_t outlen)
{
     int length = 0;
     char *p;

     p = buffer + 1;

     while (*p && (outlen > 0)) {
          if (*p == '"') {
               return length;
          }

          if (*p != '\\') {
               *(output++) = *(p++);
               outlen--;
               length++;
               continue;
          }

          switch (p[1]) {
          default:
               *(output++) = p[1];
               break;

          case 'n':
               *(output++) = '\n';
               break;
```

```
            case 'r':
                 *(output++) = '\r';
                 break;

            case 't':
                 *(output++) = '\t';
                 break;
            }

            outlen--;
            length++;
       }

       fprintf(stderr, "String is not terminated\n");
       return 0;
  }

  static int encode_data_tlv(char *buffer, char **endptr,
                     uint8_t *output, size_t outlen)
  {
       int depth = 0;
       int length;
       char *p;

       for (p = buffer; *p != '\0'; p++) {
            if (*p == '{') depth++;
            if (*p == '}') {
                 depth--;
                 if (depth == 0) break;
            }
       }

       if (*p != '}') {
            fprintf(stderr, "No trailing '}' in string starting "
                 "with \"%s\"\n",
                 buffer);
            return 0;
       }

       *endptr = p + 1;
       *p = '\0';

       p = buffer + 1;
       while (isspace((int) *p)) p++;
```

```
        length = encode_tlv(p, output, outlen);
        if (length == 0) return 0;

        return length;
    }

    static int encode_data(char *p, uint8_t *output, size_t outlen)
    {
        int length;

        if (!isspace((int) *p)) {
            fprintf(stderr, "Invalid character following attribute "
                "definition\n");
            return 0;
        }

        while (isspace((int) *p)) p++;

        if (*p == '{') {
            int sublen;
            char *q;

            length = 0;

            do {
                while (isspace((int) *p)) p++;
                if (!*p) {
                    if (length == 0) {
                        fprintf(stderr, "No data\n");
                        return 0;
                    }

                    break;
                }

                sublen = encode_data_tlv(p, &q, output, outlen);
                if (sublen == 0) return 0;

                length += sublen;
                output += sublen;
                outlen -= sublen;
                p = q;
            } while (*q);

            return length;
        }
```

```
        if (*p == '"') {
             length = encode_data_string(p, output, outlen);
             return length;
        }

        length = 0;
        while (*p) {

             char *c1, *c2;

             while (isspace((int) *p)) p++;

             if (!*p) break;

             if(!(c1 = memchr(hextab, tolower((int) p[0]), 16)) ||
                !(c2 = memchr(hextab, tolower((int)  p[1]), 16))) {
                  fprintf(stderr, "Invalid data starting at "
                       "\"%s\"\n", p);
                  return 0;
             }

             *output = ((c1 - hextab) << 4) + (c2 - hextab);
             output++;
             length++;
             p += 2;

             outlen--;
             if (outlen == 0) {
                  fprintf(stderr, "Too much data\n");
                  return 0;
             }
        }

        if (length == 0) {
             fprintf(stderr, "Empty string\n");
             return 0;
        }

        return length;
   }
```

```
   static int decode_attr(char *buffer, char **endptr)
   {
        long attr;

        attr = strtol(buffer, endptr, 10);
        if (*endptr == buffer) {
             fprintf(stderr, "No valid number found in string "
                  "starting with \"%s\"\n", buffer);
             return 0;
        }

        if (!**endptr) {
             fprintf(stderr, "Nothing follows attribute number\n");
             return 0;
        }

        if ((attr <= 0) || (attr > 256)) {
             fprintf(stderr, "Attribute number is out of valid "
                  "range\n");
             return 0;
        }

        return (int) attr;
   }

   static int decode_vendor(char *buffer, char **endptr)
   {
        long vendor;

        if (*buffer != '.') {
             fprintf(stderr, "Invalid separator before vendor id\n");
             return 0;
        }

        vendor = strtol(buffer + 1, endptr, 10);
        if (*endptr == (buffer + 1)) {
             fprintf(stderr, "No valid vendor number found\n");
             return 0;
        }

        if (!**endptr) {
             fprintf(stderr, "Nothing follows vendor number\n");
             return 0;
        }
```

```
        if ((vendor <= 0) || (vendor > (1 << 24))) {
             fprintf(stderr, "Vendor number is out of valid range\n");
             return 0;
        }

        if (**endptr != '.') {
             fprintf(stderr, "Invalid data following vendor number\n");
             return 0;
        }
        (*endptr)++;

        return (int) vendor;
   }

   static int encode_tlv(char *buffer, uint8_t *output, size_t outlen)
   {
        int attr;
        int length;
        char *p;

        attr = decode_attr(buffer, &p);
        if (attr == 0) return 0;

        output[0] = attr;
        output[1] = 2;

        if (*p == '.') {
             p++;
             length = encode_tlv(p, output + 2, outlen - 2);

        } else {
             length = encode_data(p, output + 2, outlen - 2);
        }

        if (length == 0) return 0;
        if (length > (255 - 2)) {
             fprintf(stderr, "TLV data is too long\n");
             return 0;
        }

        output[1] += length;

        return length + 2;
   }
```

```
   static int encode_vsa(char *buffer, uint8_t *output, size_t outlen)
   {
        int vendor;
        int attr;
        int length;
        char *p;

        vendor = decode_vendor(buffer, &p);
        if (vendor == 0) return 0;

        output[0] = 0;
        output[1] = (vendor >> 16) & 0xff;
        output[2] = (vendor >> 8) & 0xff;
        output[3] = vendor & 0xff;

        length = encode_tlv(p, output + 4, outlen - 4);
        if (length == 0) return 0;
        if (length > (255 - 6)) {
             fprintf(stderr, "VSA data is too long\n");
             return 0;
        }

        return length + 4;
   }

   static int encode_evs(char *buffer, uint8_t *output, size_t outlen)
   {
        int vendor;
        int attr;
        int length;
        char *p;

        vendor = decode_vendor(buffer, &p);
        if (vendor == 0) return 0;

        attr = decode_attr(p, &p);
        if (attr == 0) return 0;

        output[0] = 0;
        output[1] = (vendor >> 16) & 0xff;
        output[2] = (vendor >> 8) & 0xff;
        output[3] = vendor & 0xff;
        output[4] = attr;

        length = encode_data(p, output + 5, outlen - 5);
        if (length == 0) return 0;
```

```
        return length + 5;
    }

    static int encode_extended(char *buffer,
                       uint8_t *output, size_t outlen)
    {
        int attr;
        int length;
        char *p;

        attr = decode_attr(buffer, &p);
        if (attr == 0) return 0;

        output[0] = attr;

        if (attr == 26) {
             length = encode_evs(p, output + 1, outlen - 1);
        } else {
             length = encode_data(p, output + 1, outlen - 1);
        }
        if (length == 0) return 0;
        if (length > (255 - 3)) {
             fprintf(stderr, "Extended Attr data is too long\n");

             return 0;
        }

        return length + 1;
    }

    static int encode_extended_flags(char *buffer,
                         uint8_t *output, size_t outlen)
    {
        int attr;
        int length, total;
        char *p;

        attr = decode_attr(buffer, &p);
        if (attr == 0) return 0;

        /* output[0] is the extended attribute */
        output[1] = 4;
        output[2] = attr;
        output[3] = 0;
```

```
        if (attr == 26) {
             length = encode_evs(p, output + 4, outlen - 4);
             if (length == 0) return 0;

             output[1] += 5;
             length -= 5;
        } else {
             length = encode_data(p, output + 4, outlen - 4);
        }
        if (length == 0) return 0;

        total = 0;
        while (1) {
             int sublen = 255 - output[1];

             if (length <= sublen) {
                  output[1] += length;
                  total += output[1];
                  break;
             }

             length -= sublen;

             memmove(output + 255 + 4, output + 255, length);
             memcpy(output + 255, output, 4);

             output[1] = 255;

             output[3] |= 0x80;

             output += 255;
             output[1] = 4;
             total += 255;
        }

        return total;
   }

   static int encode_rfc(char *buffer, uint8_t *output, size_t outlen)
   {
        int attr;
        int length, sublen;
        char *p;

        attr = decode_attr(buffer, &p);
        if (attr == 0) return 0;
```

```
        length = 2;
        output[0] = attr;
        output[1] = 2;

        if (attr == 26) {
             sublen = encode_vsa(p, output + 2, outlen - 2);

        } else if ((*p == ' ') || ((attr < 241) || (attr > 246))) {
             sublen = encode_data(p, output + 2, outlen - 2);

        } else {
             if (*p != '.') {
                  fprintf(stderr, "Invalid data following "
                       "attribute number\n");
                  return 0;
             }

             if (attr < 245) {
                  sublen = encode_extended(p + 1,
                                  output + 2, outlen - 2);
             } else {

                  /*
                   *   Not like the others!
                   */
                  return encode_extended_flags(p + 1, output, outlen);
             }
        }
        if (sublen == 0) return 0;

        if (sublen > (255 -2)) {
             fprintf(stderr, "RFC Data is too long\n");
             return 0;
        }

        output[1] += sublen;
        return length + sublen;
   }

   int main(int argc, char *argv[])
   {
        int lineno;
        size_t i, outlen;
        FILE *fp;
        char input[8192], buffer[8192];
        uint8_t output[4096];
```

```
        if ((argc < 2) || (strcmp(argv[1], "-") == 0)) {
             fp = stdin;
        } else {
             fp = fopen(argv[1], "r");
             if (!fp) {
                  fprintf(stderr, "Error opening %s: %s\n",
                       argv[1], strerror(errno));
                  exit(1);
             }
        }

        lineno = 0;
        while (fgets(buffer, sizeof(buffer), fp) != NULL) {
             char *p = strchr(buffer, '\n');

             lineno++;

             if (!p) {
                  if (!feof(fp)) {
                       fprintf(stderr, "Line %d too long in %s\n",
                            lineno, argv[1]);
                       exit(1);
                  }
             } else {
                  *p = '\0';
             }

             p = strchr(buffer, '#');
             if (p) *p = '\0';

             p = buffer;

             while (isspace((int) *p)) p++;
             if (!*p) continue;

             strcpy(input, p);
             outlen = encode_rfc(input, output, sizeof(output));
             if (outlen == 0) {
                  fprintf(stderr, "Parse error in line %d of %s\n",
                       lineno, input);
                  exit(1);
             }

             printf("%s -> ", buffer);
             for (i = 0; i < outlen; i++) {
                  printf("%02x ", output[i]);
             }
```

```
            printf("\n");
        }

        if (fp != stdin) fclose(fp);

        return 0;
    }
    ------------------------------------------------------------
```

Authors' Addresses

Alan DeKok
Network RADIUS SARL
57bis blvd des Alpes
38240 Meylan
France

EMail: aland@networkradius.com
URI: http://networkradius.com

Avi Lior

EMail: avi.ietf@lior.org