Network Working Group   Alan DeKok
INTERNET-DRAFT   Network RADIUS
Category: Proposed Standard   Avi Lior
Updates: 2865, 2866, 5176   BWS
<draft-ietf-radext-radius-extensions-00.txt>
Expires: August 27, 2011
27 February 2011

Remote Authentication Dial In User Service (RADIUS) Protocol Extensions
draft-ietf-radext-radius-extensions-00.txt

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 which can carry more than 253 octets of data. This document defines changes to RADIUS which address all of the above problems.

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 imnpractical, and a more powerful mechanism is needed. Multiple attributes have been defined which 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 modifications to RADIUS:

* 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.

* allocating 4 attributes as using the format of "Extended Type". This allocation extends the RADIUS Attribute Type Space by approximately 1000 values.

* defining an "Extended Type with Flags" format, which inserts an additional "Flags" octet between the "Extended Type" octet, and the "Value" field. This method gives us a general way of adding additional functionality to the protocol.

* defining method which uses the "Flags" field to indicate data fragmentation across multiple Attributes. This method provides a standard way for an Attribute to carry more than 253 octets of data.

* allocating 2 attributes as using the format of "Extended Type with Flags". This allocation extends the RADIUS Attribute Type Space by an additional 500 values.

* defining a new "Type Length Value" (TLV) data type. The 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.

* defining a new "extended Vendor-Specific" (EVS) data type. The data type allows an attribute to carry Vendor-Specific Attributes (VSAs) inside of the new attribute formats.

* allocating 6 attributes using the new EVS data type. This allocation extends the Vendor-Specific Attribute type space by over 1500 values.

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 implementation difficulty.

1.1. 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). However, the Value field of the attribute does not follow the format required by the data type defined for that Attribute. e.g. an Attribute of type "address" which encapsulates more than four, or less than four, octets of data.

1.2. 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].

An implementation is not compliant if it fails to satisfy one or more of the must or must not requirements for the protocols it implements. An implementation that satisfies all the MUST, MUST NOT, SHOULD, and SHOULD NOT requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST and MUST NOT requirements but not all the SHOULD or SHOULD NOT requirements for its protocols is said to be "conditionally compliant".

2. Extensions to RADIUS

This section defines two new attribute formats; "Extended Type"; and "Extended Type with Flags". The formats are defined below.

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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

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

Length

This field is identical to the Length field of the Attribute format defined in [RFC2865] Section 5. 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 deemed to be an "invalid attribute", it SHOULD be silently discarded. If it is not discarded, it MUST NOT be handled in the same manner as a well-formed attribute.

Note that an "invalid attribute" does not cause the entire packet to be discarded, or to be treated as a negative acknowledgement. Instead, only the "invalid attribute" is discarded.

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified by IANA. 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 SHOULD 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 2.5, 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 SHOULD be a valid RADIUS data type.

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 "Extended Type with flags" format should 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. Extended Type with Flags

This section defines a new attribute format, called "Extended Type with Flags". 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |M|   Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

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

Length

This field is identical to the Length field of the Attribute format defined in [RFC2865] Section 5. Permitted values are between 5 and 255. If a client or server receives an "Extended

Attribute with Flags" with a Length of 2, 3, or 4, then that Attribute MUST be deemed to be an "invalid attribute", it SHOULD be silently discarded. If it is not discarded, it MUST NOT be handled in the same manner as a well-formed attribute.

Note that an "invalid attribute" does not cause the entire packet to be discarded, or to be treated as a negative acknowledgement. Instead, only the "invalid attribute" is discarded.

Extended-Type

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

M (More)

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

If the More flag is set (1), it indicates that the Value field has been fragmented across multiple RADIUS attributes. When the More flag is set (1), the attribute SHOULD have a Length field of value 255; it MUST NOT have a length Field of of value 4; 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 flag set (1).

When the Length field of an attribute has value less than 255, the More flag SHOULD be clear (0).

If a client or server receives an attribute fragment with the "More" flag set (1), but for which no subsequent fragment can be found, then the fragmented attribute is deemed to be an "invalid attribute" and the entire set of fragments SHOULD be silently discarded. If the fragmented attribute is not discarded, it MUST NOT be handled in the same manner as a well-formed attribute.

Flags

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

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 value less than 255), or it may contain a fragment of data (when the More field is set).

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.

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

2.3. TLV Data Type

We define a new data type in RADIUS, called "tlv". The "tlv" data type is an encapsulation layer which which 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 re-uses the RADIUS attribute format, as given below:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   TLV-Type    |  TLV-Length   |     TLV-Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TLV-Type

The Type field is one octet. Up-to-date values of this field are specified by IANA. Values of zero (0) MUST NOT be used. Values 254-255 are "Reserved" for use by future extensions to RADIUS. The value 26 has no special meaning.

As with Extended-Type above, the TLV-Type is meaningful only within a 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".

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 which encapsulates that TLV MUST be deemed to be an "invalid attribute", it SHOULD be silently discarded. If the encapsulating attribute is not discarded, it MUST NOT be handled in the same manner as a well-formed attribute.

Note that an "invalid attribute" does not cause the entire packet to be discarded, or to be treated as a negative acknowledgement. Instead, only the "invalid attribute" is discarded.

TLV-Value

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

The TLV-Value field SHOULD encapsulate a previously defined RADIUS data type. Using non-standard data types is NOT RECOMMENDED. We note that the TLV-Value field MAY also contain one or more attributes of data type "tlv", which allows for simple grouping and multiple layers of nesting.

The TLV-Value field is limited to containing 253 or fewer octets of data. Specifications which require a TLV to contain more than 253 octets of data are incompatible with RADIUS, and need to be redesigned. Specifications which require the transport of empty Values (i.e. Length = 2) are incomaptible 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. We RECOMMEND using type "tlv" for VSAs, in preference to any other format.

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 over-fill the "container" TLV, the "container" TLV MUST be considered to be an "invalid attribute", and handled as described above.

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.

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 which which permits the "Value" field of an Attribute to contain a Vendor-Id, followed by a Vendor-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 use in attributes which carry Vendor-Specific information, as is done with the Vendor-Specific Attribute (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 "Extended Type with Flags". 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Vendor-Id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Vendor-Type   |  String ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Vendor-Id

The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code 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.

String

The String field is one or more octets. It SHOULD encapsulate a previously defined RADIUS data type. Using non-standard data types is NOT RECOMMENDED. We note that the String 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. We recognise that Vendors have complete control over the contents and format of the String field, while at the same time recommending 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 "String" field is implicit, and is determined by taking the "Length" of the encapsulating RADIUS Attribute, and subtracting the attribute overhead (3, or 4 octets).

2.5. Attribute Naming and Type Identifiers

Attributes have traditionally been identified by a unique name and number. For example, the attribute named "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.

2.5.1. 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 use their company name as the prefix for VSA names, though this practice is not always used. For example, the name "Vendor-My-Attribute" is preferred over the name "My-Attribute". The second form can conflict with attributes from other vendors, whereas the first form cannot.

We therefore RECOMMEND that specifications give names to Attributes which attempt to be globally unique across all RADIUS Attributes. We RECOMMEND that vendors use their name as a unique prefix for attribute names. We recognise that these suggestion 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 named "Time-Zone", "Time-Day", "Time-Hour", "Time-Minute", etc.

2.5.2. 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, and 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".

The algorithm used to create the Attribute Identifier is simply to concatenate all of the various identification fields (e.g. Type, Extended-Type, etc.), starting from the encapsulating attribute, down to the final encapsulated TLV, separated by a '.' character.

2.5.3. 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 replied 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, wth TLV-Type number four (4).

2.5.4. 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 24-bit Vendor-Id, and possibly additional VSAs. Where the VSAs follow the [RFC2865] Section 5.26 recommended format, 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.

Note that there is no restriction on the size of the numerical values

in this notation. The Vendor-Id is a 24-bit number, and the VSA may have been assigned from a 16-bit vendor-specific Attribute type space.

For example, the company USR has been allocated Vendor-Id 429, and has defined a "Version-Id" attribute as number 32768. This VSA can be uniquely identified as 26.429.32768.

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

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 "Extended Type with Flags", 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   Extended-Type-Flagged-1
246   Extended-Type-Flagged-2
```

The rest of this section gives a detailed definition 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 by IANA, in the 241.{1-255} RADIUS Attribute Type Space. Further definition of this field is given in Section 2,1, above.

String

The String field is one or more octets. Implementations not supporting this specification SHOULD support the field as undistinguished octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the String 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 by IANA, in the 242.{1-255} RADIUS Attribute Type Space. Further definition of this field is given in Section 2,1, above.

String

The String field is one or more octets. Implementations not supporting this specification SHOULD support the field as undistinguished octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the String 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 by IANA, in the 243.{1-255} RADIUS Attribute

Type Space. Further definition of this field is given in Section 2,1, above.

String

The String field is one or more octets. Implementations not supporting this specification SHOULD support the field as undistinguished octets.

Implementations supporting this specification MUST use the Identifier of "Type.Extended-Type" to determine the interpretation of the String 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 by IANA, in the 244.{1-255} RADIUS Attribute Type Space. Further definition of this field is given in Section 2,1, above.

String

The String field is one or more octets. Implementations not supporting this specification SHOULD support the field as

undistinguished octets.

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

3.5. Extended-Type-Flagged-1

Description

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

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |M|   Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

245 for Extended-Type-Flagged-1

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified by IANA, in the 245.{1-255} RADIUS Attribute Type Space. Further definition of this field is given in Section 2,1, above.

M (More)

The More Flag 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.

Flags

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.

String

The String field is one or more octets. Implementations not supporting this specification SHOULD support the field as undistinguished octets.

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

3.6. Extended-Type-Flagged-2

Description

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

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |M|   Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

246 for Extended-Type-Flagged-2

Length

>= 4

Extended-Type

The Extended-Type field is one octet. Up-to-date values of this field are specified by IANA, in the 246.{1-255} RADIUS Attribute Type Space. Further definition of this field is given in Section 2,1, above.

M (More)

The More Flag 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.

Flags

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.

String

The String field is one or more octets. Implementations not supporting this specification SHOULD support the field as undistinguished octets.

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

4. Vendor Specific Attributes

We define six new attributes which 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 of "Extended Type with Flags" format, with Type codes (245.26, 246.26), using 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 a detailed definition 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Vendor-Id ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ... Vendor-Id  (cont)                        |  Vendor-Type  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  String ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type.Extended-Type

241.26 for Extended-Vendor-Specific-1

Length

>= 9

Vendor-Id

The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code 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.

String

The String 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 String 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Vendor-Id ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ... Vendor-Id  (cont)                        |  Vendor-Type  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  String ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type.Extended-Type

242.26 for Extended-Vendor-Specific-2

Length

>= 9

Vendor-Id

The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code 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.

String

The String 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 String 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Vendor-Id ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ... Vendor-Id  (cont)                        |  Vendor-Type  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  String ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type.Extended-Type

243.26 for Extended-Vendor-Specific-3

Length

>= 9

Vendor-Id

The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code 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.

String

The String 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 String 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-3 Attribute format is shown below. The fields are transmitted from left to right.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |  Vendor-Id ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ... Vendor-Id  (cont)                        |  Vendor-Type  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  String ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type.Extended-Type

244.26 for Extended-Vendor-Specific-4

Length

>= 9

Vendor-Id

The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code 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.

String

The String 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 String 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |M|   Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Vendor-Id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Vendor-Type  |  String ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 String field only.

M (More)

The More Flag 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.

Flags

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 high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code 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.

String

The String 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 String 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Extended-Type |M|   Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|                            Vendor-Id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Vendor-Type   |  String ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 String field only.

M (More)

The More Flag 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.

Flags

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 high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code 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.

String

The String 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 String field.

5. Compatibility with traditional RADIUS

There are a number of potential compatibility issues with traditional RADIUS. This section describes them.

5.1. Attribute Allocation

Some vendors have used Attribute Type codes from the "Reserved" space, as Vendor Specific Attributes. This practice is considered anti-social behavior, as noted in [GUIDELINES]. 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 which indicates which type of attributes are being sent from the client. If the flag is set one way, the conflicting attributes can be interpreted as being improperly defined Vendor Specific Attributes. If the flag is set the other way, 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 re-use the above flag to determine which type 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.

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 implementors 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 which they do not understand (i.e. attributes which have no "dictionary" definitions).

This practice is NOT RECOMMENDED. Proxy servers SHOULD forward attributes, even ones which they do not understand, or which are not in a local dictionary. When forwarded, these attributes SHOULD be sent verbatim, with no modifications or changes. 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 which 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 in [EDUROAM] many proxies do not follow these practices for unknown Attributes. Some proxies filter out unknown attributes or attributes which 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), with the rest (63%, 44/70) following 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 which 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

We recommend the following guidelines when designing 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.

* Unless otherwise specified, the guidelines in [GUIDELINES] MUST be followed.

* The data type "esv" 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. While its use is NOT RECOMMENDED by [GUIDELINES], this specification updates [GUIDELINES] to permit the "tlv" data type in attributes using the Extended-Type format.

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

6.1. 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.

* When the Type spaces of 241.*, 242.*, 243.*, or 244.* are nearing exhaustion, a new specification SHOULD be written which requests allocation of one or more RADIUS Attributes from the "Reserved" space, using the "Extended Type" format. This process is preferable to allocating "small" attributes from the 256.* and 246.* Type spaces.

* When the Type spaces of 245.* or 246.* are nearing exhaustion, a new specification SHOULD be written which requests allocation of one or more RADIUS Attributes from the "Reserved" space, using the "Extended Type with flags" format.

* All other specifications SHOULD NOT request allocation from the standard Attribute Type Space (i.e. Attributes 1 through 255). That space is deprecated, and is not to be used.

* Attributes which encode 252 octets or less of data SHOULD

request allocation from the Type spaces of 241.*, 242.*, 243.*, or 244.*.

* Attributes which encode 253 octets or more of data MUST request allocation from the Type spaces of 245.* or 246.*.

* Where a group of TLVs is strictly defined, and not expected to change, and and totals less than 247 octets of data, they SHOULD request allocation from the Type spaces of 241.*, 242.*, 243.*, or 244.*.

* Where a group of TLVs is loosely defined, or is expected to change, they SHOULD request allocation from the Type spaces of 245.* or 246.*.

6.2. TLV Guidelines

The following items give 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.

* Specifications SHOULD that the interpretation of an attribute depends only on its OID, and not on its encoding in the RADIUS packet.

6.3. Implementation Guidelines

* RADIUS Server implementations SHOULD support this specification as soon as possible.

* RADIUS Proxy servers SHOULD forward all attributes, even ones which they do not understand, or which are not in a local dictionary. These attributes SHOULD be forwarded verbatim, with no modifications or changes.

* Any attribute which is allocated from the Type spaces of 245.* or 246.*, of data type "text", "string", or "tlv" can end up carrying more than 251 octets of data, up to the maximum RADIUS packet length (~4096 octets). Specifications defining such attributes SHOULD define a maximum length.

6.4. 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 be widely deployed.

7. Rationale

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 to be unworkable in practice. We do not otherwise explain here why earlier proposals did not obtain working group consensus.

This proposal has the benefit of being simple, as the "Extended Type" format requires only a one octet change to the Attribute format.

7.1. Attribute Audit

An audit of almost five thousand publicly available attributes [ATTR], 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
235	string
96	other data types
35	IPv6 Address
18	date
4	Interface Id
3	IPv6 Prefix
4683	Total

The entries in the "Data Type" column are data types recommended by [GUIDELINES]. The "other data types" row encompasses data types not recommended by that document.

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. The "Extended Type with Flags"

formats is therefore important, but "long" attributes have had limited deployment.

8. 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, only attributes.

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

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

HEXCHAR = HEXDIG HEXDIG

STRING = DQUOTE 1*CHAR DQUOTE

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

DATA = 1*HEXCHAR / 1*TLV / STRING

LINE = Identifier DATA
```

The progam has additional restrictions on its input that are not reflected in the above grammar. For example, the portions of the Identifier which 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 comprehensability 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 which 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 field(s); 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 which contains non-UTF8 data, or a Framed-IP-Address which contains 253 octets of ASCII data. As a result, it cannot 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", "Flags", "Vendor-Id", "Vendor-Type", and "Vendor-Length". It can therefore be used to encode example attributes from input which is humanly readable.

We do not give examples of "malformed" or "invalid attributes". We also note that the examples show format, and not 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.

8.1. Extended Type

The following are 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
```

8.2. Extended Type with Flags

The following are a series of examples of the "Extended Type with flags" 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 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        aaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
        bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
        bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
        bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
        bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbcccccccccccccccccccc
        cccccccccc
    -> 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 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
```

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 flag set, and carries the Vendor-Id and Vendor-Type. The second attribute has the More flag 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 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
            bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbcccccccccccccc
            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 bb f5 17 1a 00 bb
     bb bb bb bb cc cc cc cc cc cc cc cc cc cc 13 45 67 89
```

9. IANA Considerations

This document has multiple impacts on IANA, in the "RADIUS Attribute Types" registry. Attribute types which were previously reserved are now allocated, previously free attributes are marked deprecated, and the registry is extended from a simple 8-bit array to a tree-like structure, up to a maximum depth of 125 nodes.

9.1. Attribute Allocations

IANA is requested to move the "Unassigned" numbers in the range 144-191 from "Unassigned" to "Deprecated". This status means that allocations SHOULD NOT be made from this space. Instead, allocations SHOULD be taken from the Extended Type space, starting with lower numbered attributes. However, allocation from the "Deprecated" space MAY still be performed by publication of an IETF specification, where that specification requests allocation from the "Deprecated" space, and gives reasons why use of the Extended Type space is impossible.

IANA is requested to move the following numbers from "Reserved", to allocated, with the following names:

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

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

9.2. RADIUS Attribute Type Tree

Each of the attributes allocated above extends the "RADIUS Attribute Types" to an N-ary tree, via a "dotted number" notation. Each number in the tree is an 8-bit value (1 to 255). The value zero (0) MUST NOT be used. Currently, only one level of the tree is defined:

```
* 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
* 243          Extended-Attribute-3
* 242.{241-255} Reserved
* 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
* 245.26        Extended-Vendor-Specific-6
* 246.{27-240}  Unassigned
* 246.{241-255} Reserved
```

The values marked "Unassigned" above are available for assignment by IANA in future RADIUS specifications. The values marked "Reserved" are reserved for future use.

9.3. Assignment Policy

Attributes which are known to always require 252 octets or less of data MUST be assigned from the lowest unassigned number, e.g. 241.1, 241.2, 241.3, etc. Attributes have the potential to transport more than 252 octets of data MUST be assigned from the 245.* or 246.* spaces, again using the lowest unassigned number, and MUST request assignment from the appropriate Attribute Type Space.

The above policy can be difficult to enforce in the case of TLVs. For exaple, a set of TLVs may define a logical structure which totals less than 252 octets of data. Later extensions could assign additional sub-TLVs, and extend the structure to more than 252 octets of data. This capability means that TLV definitions SHOULD generally request assignment from the 245.* or 246.* space.

9.4. Extending the Attribute Type Tree

New specifications may request that the tree be extended to an additional level or levels. The attribute MUST be of type "tlv".

For example, a specification may request that an "Example-TLV" attribute be assigned, of data type "tlv". If it is assigned the number 245.1, then it will define an extension to the registry as follows:

```
* 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 number zero (0) MUST NOT be used. The last two numbers (254 and 255) MUST be reserved for future use. All other numbers are available for assignment by IANA.

The Attribute Type Tree can be extended multiple levels in one specification. For example, the "Example-TLV" above could contain another attribute, "Example-Nested-TLV", of type "tlv". It would define an additional extension to the registry as follows:

```
* 245.1.1           Example-Nested-TLV
* 245.1.1.{1-253}   Unassigned
* 245.1.1.{254-255} Reserved
```

This process may be continued to additional levels of nesting.

Again, this example does not define an "Example-Nested-TLV" attribute.

9.5. Extending the RADIUS Attribute Type Space

The extended RADIUS Attribute Type space may eventually approach exhaustion. When necessary, the space SHOULD be extended by publication of a specification which allocates new attributes of either the "Extended Type", or the "Extended Type with flags" format. The specification SHOULD request allocation of a specific number from the "Reserved" RADIUS Attribute type space, such as 247. The attribute(s) SHOULD be given a name which follows the naming convention used in this document. The Extended-Type value of 26 MUST be allocated to a "Vendor Specific" attribute, of data type "esv". The Extended-Type values of 241 through 255 MUST be marked as "Reserved".

IANA SHOULD allocate the attribute(s) as requested. For example, if allocation of attribute 247 is requested, the following definitions MUST be made in the specification, and allocated by IANA.

```
* 247.1          Extended-Attribute-7
* 247.{1-25}     Unassigned
* 247.26         Extended-Vendor-Specific-7
* 247.{27-240}   Unassigned
* 247.{241-255}  Reserved
```

We note,however, that the above list is an example, and we do not request or perform allocation of attribute 247 in this document.

10. 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]. RADIUS uses the MD5 hash [RFC1321] for packet authentication and attribute obfuscation. There are ongoing efforts in the IETF to analyze and address these issues for the RADIUS protocol.

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.

11. References

11.1. Normative references

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", 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.

11.2. Informative references

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

[RFC2868] Zorn, G., et al, " RADIUS Attributes for Tunnel Protocol Support", RFC 2868, June 2000.

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

[RFC5176] Chiba, M. et al., "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)", RFC 5176, January 2008.

[RFC5234] Crocker, D. (Ed.), and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, October 2005.

[GUIDELINES]
DeKok, A., and Weber, G., "RADIUS Design Guidelines", draft-ietf-radext-design-18.txt, work in progress.

[EDUROAM] Internal Eduroam testing page, data retrieved 04 August 2010.

[ATTR] http://github.com/alandekok/freeradius-server/tree/master/share/, data retrieved September 2010.

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 which 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 a grammar similar to that given in Section 8, 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) 2010 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;
  }
  -----------------------------------------------------------
```

Author's Address

Alan DeKok
Network RADIUS SARL
15 av du Granier
38240 Meylan
France

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

Avi Lior
Bridgewater Systems Corporation
303 Terry Fox Drive
Suite 100
Ottawa, Ontario K2K 3J1
Canada

Phone: +1 (613) 591-6655
Email: avi@bridgewatersystems.com
URI: http://www.bridgewatersystems.com/