Global Information Assurance Certification Paper

“A model for configurable decoding of RADIUS Accounting and Authentication messages”

Author – Stephen Nimmons

GSEC 1.4 b Option 1 – Research on Topics in Information Security

Submission Date: February 15, 2004

Contents

Table of Figures

Glossary

CHAP	Challenge Handshake Authentication Protocol
IETF	Internet Engineering Task Force
IPX	Internetwork Packet Exchange
MD5	Message Digest 5
MIB	Management Information Base
MTU	Maximum Transmission Unit
NAS	Network Access Service
PAP	Password Authentication Protocol
PPP	Point to Point Protocol
RADIUS	Remote Authentication Dial In User Service
RFC	Request for Comment
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TLV	Type, Length, Value
UDP	User Datagram Protocol
UML	Unified Modelling Language

Abstract

The purpose of this paper is to provide an analysis of RFC2865 (Remote Authentication Dial In User Service (RADIUS) [2], [4], and RFC2866 (RADIUS Accounting) [3]. This analysis is then applied to a 'how to' design for a configurable component to process RADIUS Accounting packets. Components of this type could be used to subscribe to traffic between the NAS and the Accounting Server and to decode and maintain a lookup table of user sessions to leased IP addresses. This could be used to determine (and indeed correlate) the IP address of devices and provide an alternative method to querying LDAP [10] repositories on the RADIUS servers for such information [9]. Such a component could be used in various scenarios, such as on mobile networks where mobile devices are temporarily leased IP addresses for various value added services (such as WAP and MMS) [5], [6]. By actively subscribing to and decoding RADIUS Accounting packets in real time, a suitable component could track, maintain and service such requests. This could off load processing demands on the RADIUS LDAP repositories, leading to improved performance, and availability.

The first section of this paper describes the features of RADIUS authentication and accounting messages, and provides a UML [11] data model mapped from the contents of RFC2865 [2] and RFC2866 [3]. In the second part of the paper a generic TLV (Type Length Value) decoder design is discussed, along with an XML based model used to configure the decoder to decode RFC2865/2866 RADIUS messages. This paper concludes that with only minor exceptions to the rule, it is possible to model RFC2865/2866 messages for decoding by a generic type length value decoder configured through an XML representation of the protocol. This decouples the workings of the actual decoder with the semantic protocol logic, improves configurability and simplifies supporting changes to the protocol specifications (such as addition of new attributes).

The RADIUS Protocol Data Model

The contents of RFC2865 and 2866 have been mapped into UML to provide a simplified (and condensed) view of the message types exchanged between RADIUS entities (clients, proxies, authentication servers and accounting servers). This takes the form of UML class diagrams that show the structure and cardinality relationships of the RADIUS authentication messages (Access-Request, Access-Accept, Access-Reject and Access-Challenge), and RADIUS accounting messages (Accounting-Request and Accounting-Response). The attribute payload of each message (which contains attributes in Type Length Value (TLV) format) has been modelled using classes for each attribute (with each class containing 3 attributes, representing the type, length, value associated with each attribute in the native RADIUS encoding). This leads to a representation of the data model providing a simple reference in terms of understanding each message payload, and the multiplicity relationships each attribute has in each message type.

Overview of the RADIUS Data Format

The format of RADIUS messages is fairly straight forward, and the format depicted below (Figure 1 RADIUS Message Format) is used for all of the RADIUS authentication and accounting messages. In terms of describing the UML data model, the convention in this paper is to depict the RADIUS message in two parts. Part one is the common 'header' which (for every authentication and accounting message, contains Code, Identifier, Length and Authenticator fields (The diagram in Figure 1 is a slightly modified version of the Packet Format diagram in section 3 of RFC2865 [2]).

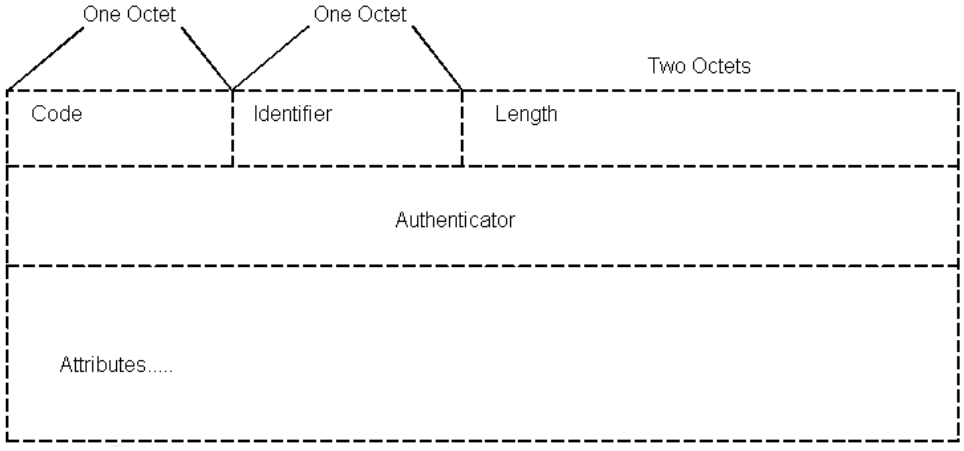

Figure 1 RADIUS Message Format

RADIUS Message Header

Code attribute

The code attribute is one octet in length and identifies the RADIUS message type. The following values are valid for the code attribute.

1	Access-Request
2	Access-Accept
3	Access-Reject
4	Accounting-Request
5	Accounting-Response
11	Access-Challenge
12	Status-Server (experimental)
13	Status-Client (experimental)
255	Reserved

In terms of RFC2865 the primary interest is in Access-Request, Access-Accept, Access-Reject, and Access-Challenge messages. RADIUS accounting messages (as defined in RFC2866) will be identified using the Code values of 4 and 5 (Accounting-Request and Accounting-Response respectively). This covers the complete set of RADIUS authentication and accounting messages. The suite of message types and operations is therefore small (4 message types for authentication, and 2 message types for accounting), but message complexity especially in Access-Request, Access-Accept and Accounting-Request should not be underestimated.

Identifier Attribute

The identifier is one octet in length, and is used by the server to match request / reply pairs.

Length Attribute

The length attribute is a two-octet value that is the length of the RADIUS packet including Code (one octet), Identifier (one octet), Length (two octets), Authenticator (16 octets) and Attribute fields (variable). Therefore the length of the variable attribute section of the packet is (Length – (20 octets)). The minimum allowed value for length is 20 octets (indicating that no attributes are present) and maximum length is 4096 octets. In client to server RADIUS communication, exactly one RADIUS packet is encapsulated in the UDP data field (sent over port 1812), and therefore determining where RADIUS 'events' start and end is trivial. Similarly NAS to Accounting Server communication is over UDP port 1813 and encapsulates single messages in UDP datagrams.

Authenticator Attribute

The authenticator is a 16 octet value (most significant octet transmitted first). This is either a Request Authenticator (as used in an Access-Request) or a Response Authenticator (as used in an Access-Accept, Access-Reject or Access-Challenge). This value is an MD5 digest [8].

RADIUS Message Body

Attributes

The remaining (variable length) part of the RADIUS packet contains a series of attributes, which are in TLV (type, length, value) format. There are a few exceptions to this rule, and this will be further explored (see "Exceptions to the TLV Rule").

The UML model for an Access-Request (as per RFC2865) is depicted in Figure 2 RADIUS Access-Request class diagram. The Access-Request header class in the UML class diagram represents the 'first section' of the RADIUS packet (containing the Code, Identifier, Length and Authenticator attributes, common to all RADIUS authentication and accounting packets). Each 'sub class' represents an attribute that occurs within the specific message. Figure 2 RADIUS Access-Request class diagram shows all of the attributes that may exist within a RADIUS Access-Request message. As these attributes have type, length, value attributes themselves, they have been modelled as classes with three attributes. The multiplicity of each attribute to the parent class, represents the valid occurrences of each attribute in each RADIUS message. For example in an Access-Request message, the USER_NAME attribute may occur zero or one times but the STATE attribute can occur 0 or many times.

All attribute type values have been supplied in the respective class diagrams. Where length is a fixed value, this has also been shown in the UML. Where length is not a fixed value, it has not been depicted, and all variable length attributes should be checked against the specifications in RFC2865/2866 [2], [3]. The purpose of mapping this data model is to provide a condensed description of each message type, and the cardinality of the relationships between attributes in each RADIUS message.

A key point in terms of processing the TLV attribute section of a RADIUS packet is to understand that attributes may occur in any order (however there are some requirements for a RADIUS proxy to preserve certain attribute ordering), and a decoder component must be able to handle this. The occurrence of each attribute in each message is best understood by studying the UML class diagrams.

Authentication Messages

Access-Request

An Access-Request message is sent by a client to a RADIUS authentication server, and is structured as below:

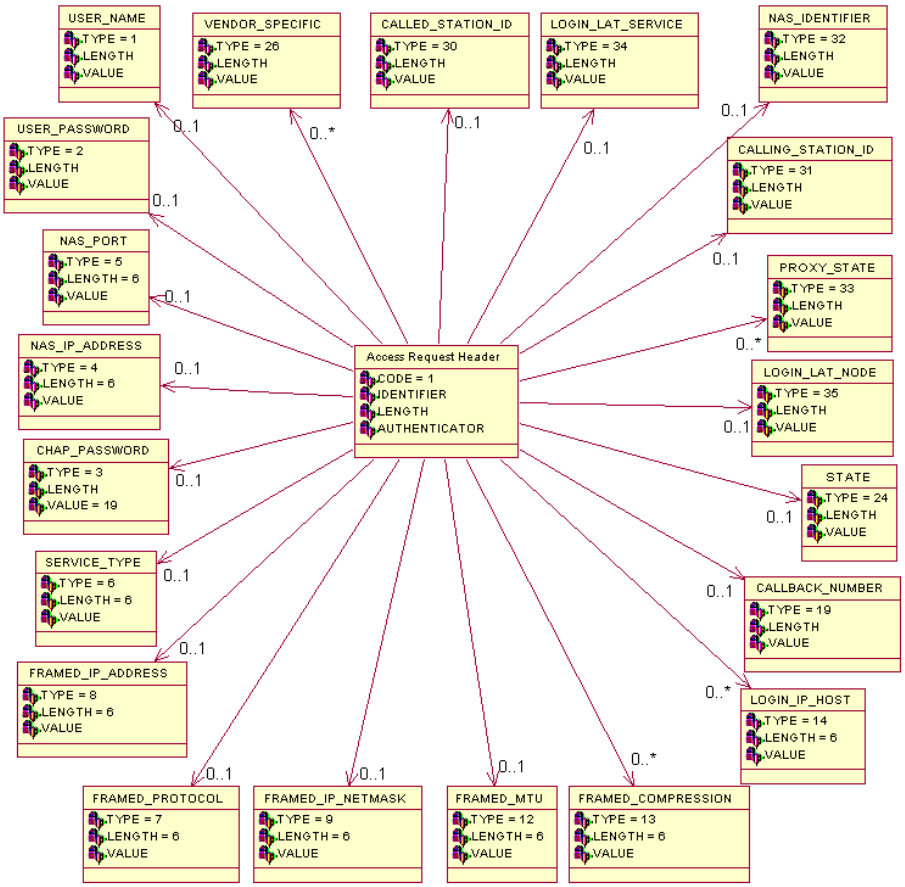

Figure 2 RADIUS Access-Request class diagram

Access-Accept

An Access-Accept message is sent from the authentication server to the client requesting the service, and is of the following format:

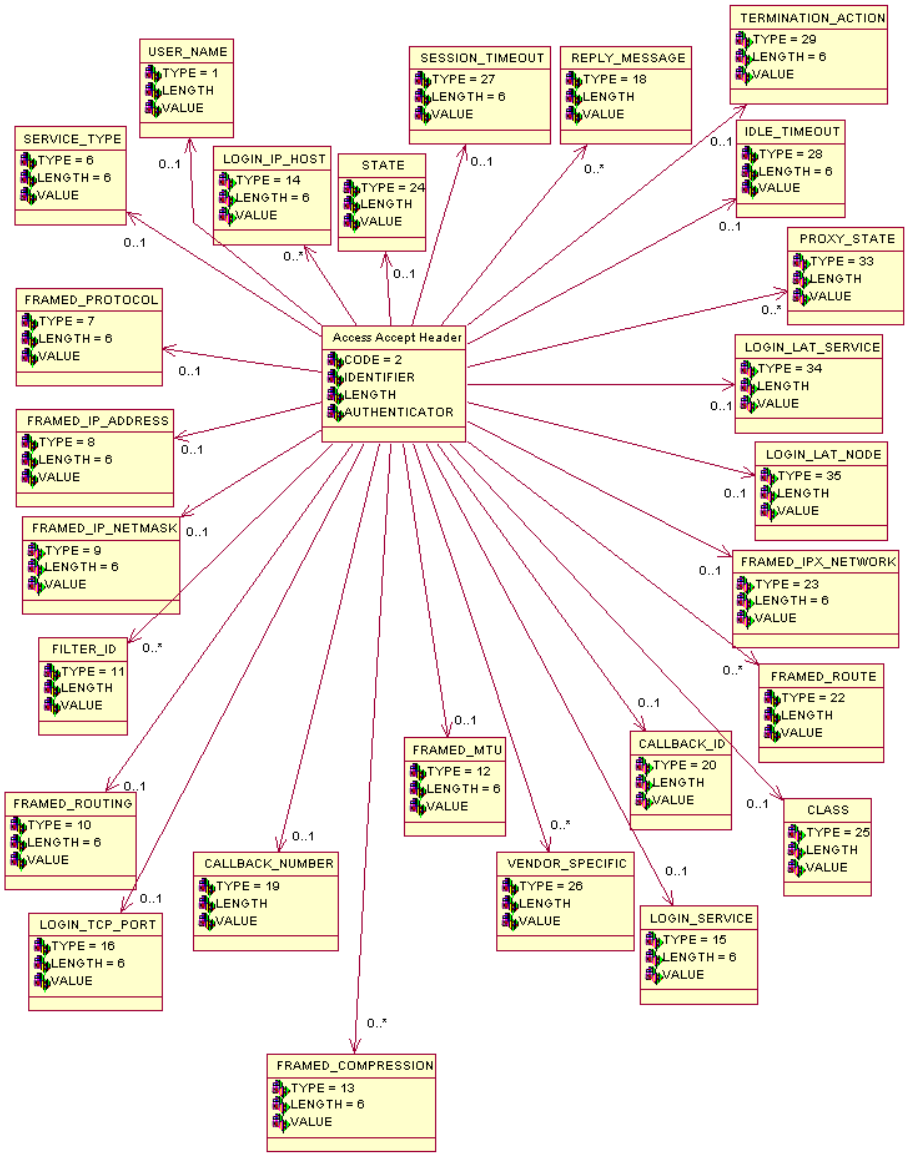

Figure 3 RADIUS Access-Accept class diagram

Access-Reject

If an authentication request fails (bad credentials, or client is requesting a proscribed service), the authentication server will respond with an Access-Reject message with the following structure:

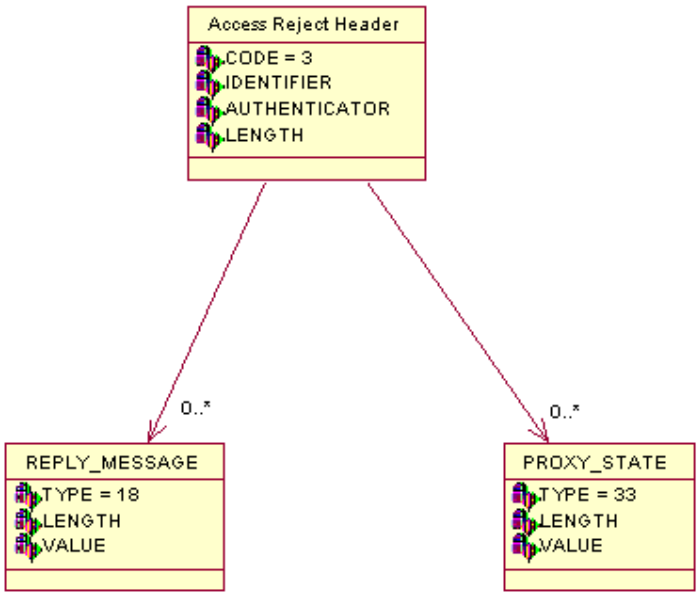

Figure 4 RADIUS Access-Reject class diagram

Access Challenge

An Access-Challenge can be periodically issued by the authentication server to the client (this will be issued after the initial authentication, during the client's session). This is simply to check that the session has not been hijacked, and the client must reply to the Access-Challenge by supplying a new Access-Request message.

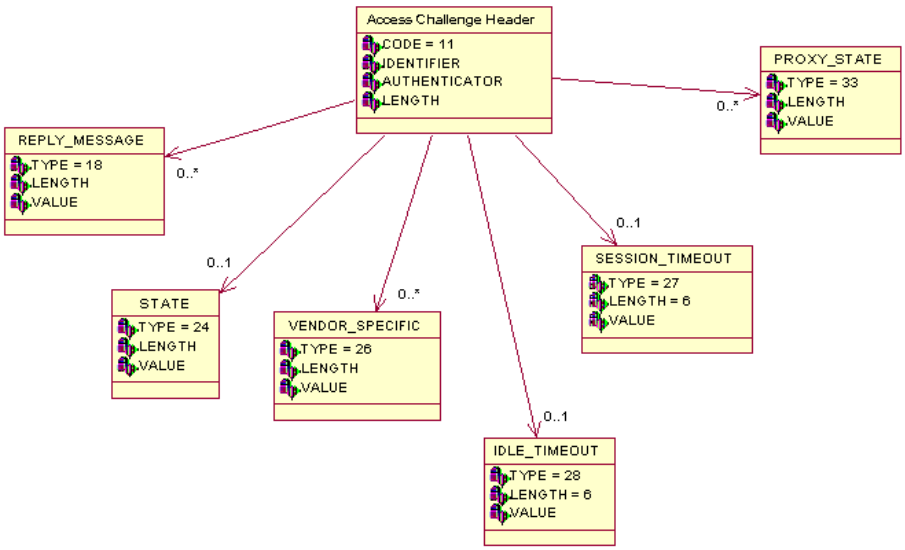

Figure 5 RADIUS Access-Challenge Class Diagram

Accounting Messages

Accounting-Request

An Accounting-Request message is sent from a NAS (i.e. the RADIUS authentication server) to the accounting server, and has a fairly complex structure (as can be seen in the UML below). The multiplicity of attributes in the message is depicted in the UML, but it is worth recapping that these attributes can occur in any order in the message. Clearly there are large numbers of permutations of valid Accounting-Request messages and to support the full RFC2866 specification, all attributes in the UML must be processed.

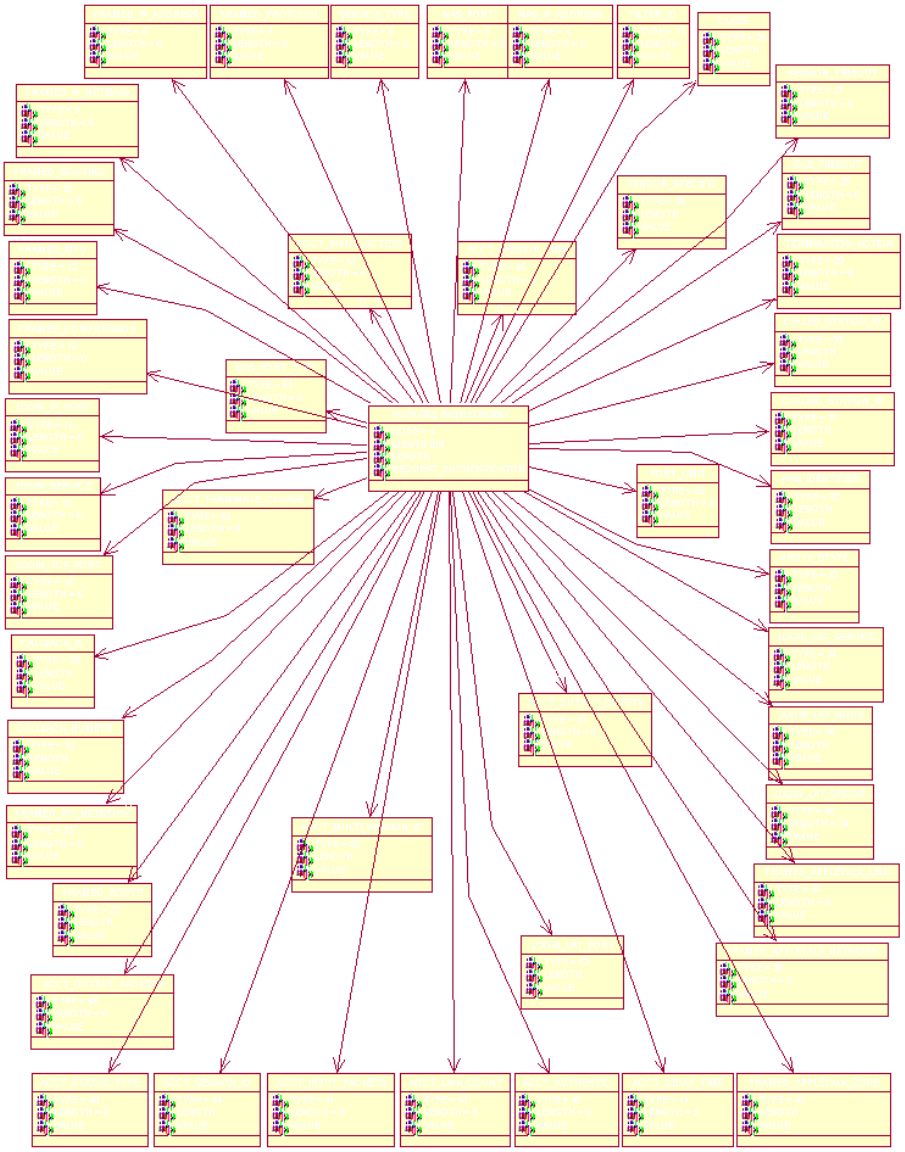

Figure 6 Accounting-Request class diagram

Accounting-Response

Accounting-Response messages are quite simple and can contain only two discrete attributes (PROXY_STATE, and VENDOR_SPECIFIC).

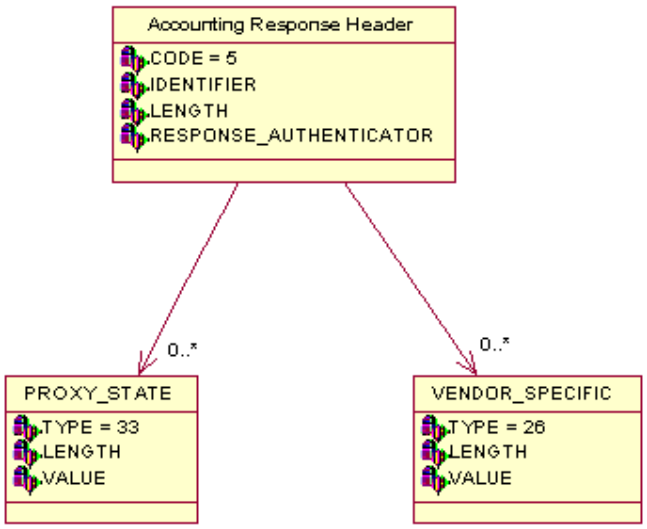

Figure 7 Accounting Response class diagram

RADIUS Protocol Event Flows

In the following examples of RADIUS Authentication and Accounting it should be understood that RADIUS is using a reliable protocol, and will resend requests and responses until they are acknowledged. An Accounting Server will not send an Accounting-Response if it cannot log the initial request, and the RADIUS daemon queues and tracks requests to trap duplicates caused by resends.

Successful Authentication

The flow in Figure 8 RADIUS Authentication / Accounting Message Flow, depicts a successful RADIUS authentication event, and the corresponding accounting start request and acknowledgement. The sequence of the flow is as follows.

1. RADIUS client sends an Access-Request packet to the RADIUS server.
2. RADIUS server successfully authenticates the client, and responds with an Access-Accept message.
3. An Accounting-Request start message is sent to the server. The Acct-Status-Type (i.e. attribute type 40) is used to indicate whether this is an accounting start or an accounting stop request. Valid values for this field are:

1	Start
2	Stop
3	Interim-Update
7	Accounting-On
8	Accounting-Off
9-14	Reserved for Tunnel Accounting
15	Reserved for Failed

There are four types of accounting packets (all are subdivisions of Accounting-Request), which differ by the value of Acct-Status-Type attribute. Typically the Session Start Packet and Session End Packet will occur.

Session Start Packet

The session start packet is sent after the user has successfully passed the authentication and has started to receive the requested service. It must contain at least following attributes:

- Acct-Status-Type = Start
- User-Name
- Acct-Session-Id
- NAS-IP-Address
- NAS-Port-Id

Session Stop Packet.

The Session Stop Packet is sent after the user has disconnected. It conveys the information about the duration of the session, number of octets transferred, etc. It must contain at least the following attributes:

- Acct-Status-Type = Stop
- User-Name
- NAS-IP-Address
- Acct-Session-Id

Accounting Off Packet

This is not a session-based packet, but a NAS sends this type of accounting request to indicate to the RADIUS server that all sessions associated with this NAS should be ended. This message would be sent when a NAS is being 'gracefully' shut down.

The packet must contain at least the following attributes:

- Acct-Status-Type = Accounting-Off
- NAS-IP-Address

Accounting On Packet

Essentially a reversal of the Accounting Off packet (sent after NAS start-up), to indicate readiness to accept sessions.

It must contain at least the following attributes:

- Acct-Status-Type = Accounting-Off
- NAS-IP-Address

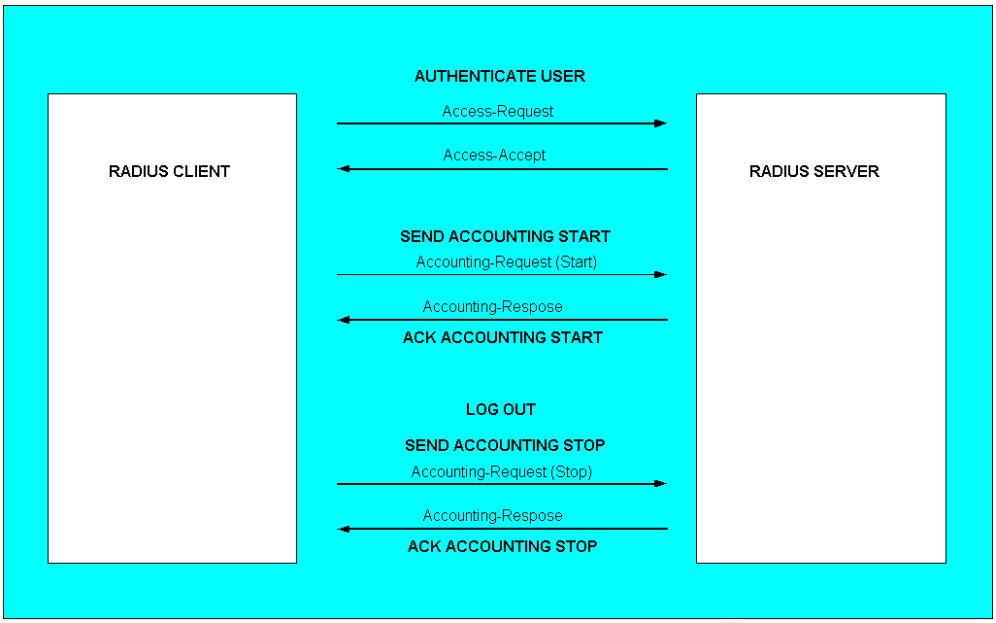

Figure 8 RADIUS Authentication / Accounting Message Flow

Failed Authentication

The simplest flow is concerned with failed authentication, which will occur as a result of bad credentials, or attempting to access a proscribed service. As the client session never authenticates, there is no corresponding Accounting-Request / Accounting-Response.

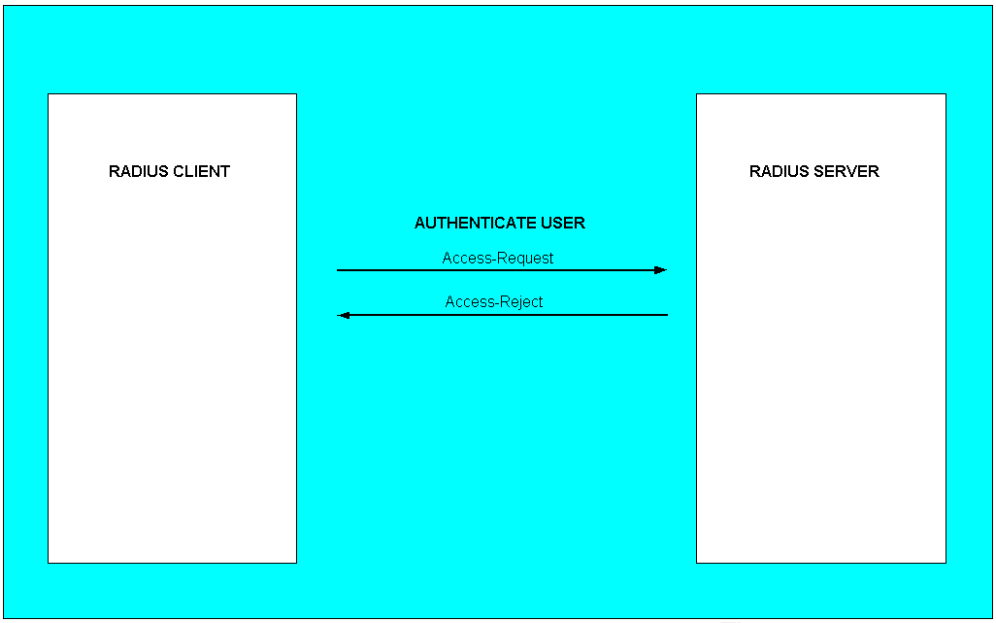

Figure 9 RADIUS Authentication Failure

RADIUS Access-Challenge

An Access-Challenge message structure has already been discussed, and an authenticated client when issued with an Access-Challenge must respond with a new Access-Request message (actually the client must return encrypted credentials, based on the challenge issued by the RADIUS server).

Design for a Generic RADIUS Message Decoder

Having discussed the RADIUS message data models and event flows in some depth, the final section of this paper presents a 'how to' approach to building a generic TLV (type / length / value) based decoder for RADIUS messages.

The suggested approach is based around modelling the RADIUS protocol in XML, and configuring a generic TLV decoder at run time with the specifications of the protocol to decode. This is achieved by using TLVProtocolAttribute objects. When these objects have associated lookup values, the mappings of decoded attribute to lookup value is stored in a HashMap. The decoded value for the attribute payload is used as the hash map key, and the corresponding string is the 'English' equivalent of the attribute's semantic value.

The XML model would be 'ingested' by the decoder at start-up. The protocol model would be cached in instantiations of TLVProtocolAttribute objects (in a HashMap, keyed on the unique attribute value as defined in RFC2865). As the TLV Decoder processes the RADIUS message, it would decode the first byte of the attribute to a lookup value. For example, for NAS_IP_ADDRESS the value would be 4, for the USER_NAME attribute the lookup value would be 1. The decoder would then recover the TLVProtocolAttribute definition of this attribute (as ingested from the XML model), and apply the implicit decoding rules based on the configuration of the protocol object. Settings in the TLVProtocolAttribute would facilitate attribute level decoding rules for:

1. Whether this attribute should be decoded
2. How this attribute should be decoded (as integer, string, ASN.1 etc.)
3. What bounds checking should be applied to the decoded result (e.g. maximum and minimum string lengths, or integer sizes).
4. How the value should be stored internally, and whether any mappings should be applied to map the decoded value onto an outbound field.

The TLV Decoder Package

The TLVDecoder package would be a generic decoder, configured via XML (a 'hypothetical class model' for this package is shown in Figure 10 Class Model for Generic Decoder). The TLVProtocolDecoder would read its XML model using the TLVProtocolDecoderConfigReader class, and store the XML model using TLVProtocolAttribute objects. Decoder exception classes would be handled via TLVProtocolDecoderException and TLVProtocolDecoderHashMapException classes.

The TLVProtocolDecoder would provide a 'generic' methodology to decode TLV encoded data (although as TLV is clearly a standard which has many variants, some changes may be needed in this model to fit 'all' possible TLV formats).

The model below has been configured to decode 'RADIUS TLV'. This demonstrates reusability and flexibility in supporting any changes in the RADIUS specification, or use of various Vendor Specific Attribute dictionaries.

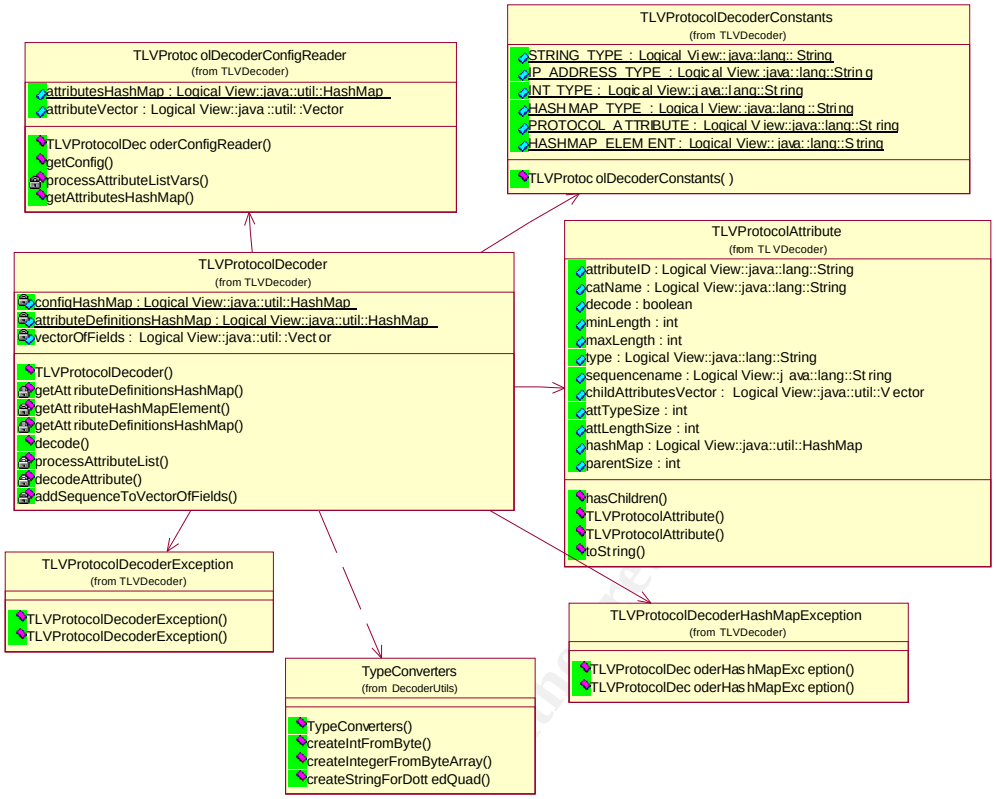

Figure 10 Class Model for Generic Decoder

XML Model to Configure Generic Decoder

The entire RADIUS protocol has been modelled in XML, as given below. This means that changes to attributes (such as addition of a new lookup value for framed_protocol) could be easily handled by changing the XML model of the protocol.

Key things to note regarding this configuration are:

- The header (Code, Identifier, Length and Authenticator should be handled separately). These attributes are not in TLV form, and would not naturally fit with the TLV decoding model suggested.

- To decode the Authenticator and CHAP password, access is required to a shared secret.

- Decoding of each attribute could be switched on or off using XML configuration (i.e. changing the decode attribute in the XML to either true or false).

- The RADIUS protocol rules for each attribute (their type, maximum and minimum lengths, lookup values etc. are encoded in XML), and could be changed through configuration.

- Mappings are exposed through XML. For example, the decoded framed protocol attribute in an inbound RADIUS Accounting-Request would be decoded and stored internally in a field called framed_protocol_field. This could be mapped onto a particular database field, object attribute or message attribute.

- The VSA (Vendor Specific Attribute) Dictionary for Netzwert has been encoded in XML in the example below. This facilitates simplified change if another RADIUS vendor were to be supported. For other dictionary formats refer to [7].

The configuration models the variable attribute section of the RADIUS message body (that is essentially a sequence of type, length, value attributes). The mark-up maps on to the structure of the TLVProtocolAttribute objects (as pictured in Figure 10 Class Model for Generic Decoder).

Understanding the Tag Structure

Tag Name	Usage	Values
Protocolattribute	Parent tag	
Protocolvalue	The integer value of this attribute once decoded	
Maptoname	If mapping to a database or outbound message this field is exposed.	
Decode	Used to switch decoding of this attribute on/off	True/false
Type	Used to indicate to the decoder the type of the payload of this attribute	String Int IP_address HashMap ASN.1 (etc.)
Minlen	The minimum length allowed for the decoded value of this attribute	
Maxlen	The maximum length allowed for the decoded length of this attribute	
Seqname	If this attribute belongs to a sequence, this would model an internal sequence name to assign	

	the decoded value	
Atttypesize	The size of the attribute type in bytes	Always 1 (1 byte) for RADIUS
Attlengthsize	The size of the attribute length identifier in bytes	Always 1 (1 byte) for RADIUS
Parentsize	Needed for the Vendor Specific attribute, as it is not a pure TLV attribute.	

A Proposed XML Model to configure a generic TLV Decoder to decode RADIUS packets

```
- <TLVDecoderConfig>

    Configured for RADIUS RFC2865 / 2866
     -->
   - <tlvattributedefs>
      - <!--
      USERNAME ATTRIBUTE
       -->
       <protocolattribute protocolvalue="1" maptoname="user_name_field" decode="true" type="string"
            minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
      - <!--
      USERPASSWORD ATTRIBUTE
       -->
       <protocolattribute protocolvalue="2" maptoname="user_password_field" decode="false" type="string"
            minlen="16" maxlen="128" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
      - <!--
      CHAPPASSWORD ATTRIBUTE
       -->
       <protocolattribute protocolvalue="3" maptoname="chap_password_field" decode="false"
            type="string" minlen="16" maxlen="16" seqname="none" atttypesize="1" attlengthsize="1"
            parentsize="0" />
      - <!--
      NASIPADDRESS ATTRIBUTE
       -->
       <protocolattribute protocolvalue="4" maptoname="nas_ip_address_field" decode="true"
            type="ip_address" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
            parentsize="0" />
      - <!--
      NASPORT ATTRIBUTE
       -->
       <protocolattribute protocolvalue="5" maptoname="nas_port_field" decode="true" type="int" minlen="4"
            maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
      - <!--
       SERVICETYPE ATTRIBUTE
       -->
      - <protocolattribute protocolvalue="6" maptoname="service_type_field" decode="true" type="hashmap"
            minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0">
         - <hashmap>
            - <!--
             The mappings of the Service Type Value Attribute to the string value output in the RV
               message
             -->
             <hashmapelement id="1" value="Login" />
             <hashmapelement id="2" value="Framed" />
             <hashmapelement id="3" value="Callback Login" />
             <hashmapelement id="4" value="Callback Framed" />
             <hashmapelement id="5" value="Outbound" />
             <hashmapelement id="6" value="Administrative" />
             <hashmapelement id="7" value="NAS Prompt" />
             <hashmapelement id="8" value="Authenticate Only" />
```

```
        <hashmapelement id="9" value="Callback NAS Prompt" />
        <hashmapelement id="10" value="Call Check" />
        <hashmapelement id="11" value="Callback Administrative" />
      </hashmap>
    </protocolattribute>
- <!--
  FRAMEDPROTOCOL ATTRIBUTE
  -->
- <protocolattribute protocolvalue="7" maptoname="framed_protocol_field" decode="true"
      type="hashmap" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0">
    - <hashmap>
        - <!--
          The mappings of the Framed Protocol Value Attribute to the string value output
          -->
        <hashmapelement id="1" value="PPP" />
        <hashmapelement id="2" value="SLIP" />
        <hashmapelement id="3" value="AppleTalk Remote Access Protocol (ARAP)" />
        <hashmapelement id="4" value="Gandalf proprietary SingleLink/MultiLink protocol" />
        <hashmapelement id="5" value="Xylogics proprietary IPX/SLIP" />
        <hashmapelement id="6" value="X.75 Synchronous" />
        <hashmapelement id="7" value="GPRS PDP Context" />
      </hashmap>
    </protocolattribute>
- <!--
  FRAMEDIPADDRESS ATTRIBUTE
  -->
  <protocolattribute protocolvalue="8" maptoname="framed_ip_address_field" decode="true"
      type="ip_address" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
  FRAMEDIPNETMASK
  -->
  <protocolattribute protocolvalue="9" maptoname="framed_ip_netmask_field" decode="true"
      type="ip_address" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
  FRAMEDROUTING
  -->
- <protocolattribute protocolvalue="10" maptoname="framed_routing_field" decode="true"
      type="hashmap" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0">
    - <hashmap>
        - <!--
          The mappings of the Framed Routing Value Attribute to the string value output
          -->
        <hashmapelement id="0" value="None" />
        <hashmapelement id="1" value="Send routing packets" />
        <hashmapelement id="2" value="Listen for routing packets" />
        <hashmapelement id="3" value="Send and Listen" />
      </hashmap>
    </protocolattribute>
- <!--
  FILTERID
  -->
  <protocolattribute protocolvalue="11" maptoname="filter_id_field" decode="true" type="string"
      minlen="1" maxlen="253" seqname="filter_id_sequence" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
  FRAMEDMTU
  -->
  <protocolattribute protocolvalue="12" maptoname="framed_mtu_field" decode="true" type="int"
      minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
  FRAMEDCOMPRESSION
  -->
- <protocolattribute protocolvalue="13" maptoname="framed_compression_field" decode="true"
      type="hashmap" minlen="4" maxlen="4" seqname="framed_compression_sequence"
      atttypesize="1" attlengthsize="1" parentsize="0">
    - <hashmap>
        - <!--
          The mappings of the Framed Compression Value Attribute to the string value output
```

```
            -->
            <hashmapelement id="0" value="None" />
            <hashmapelement id="1" value="VJ TCP/IP header compression" />
            <hashmapelement id="2" value="IPX header compression" />
            <hashmapelement id="3" value="Stac-LZS compression" />
        </hashmap>
    </protocolattribute>
- <!--
    LOGINIPHOST
    -->
    <protocolattribute protocolvalue="14" maptoname="login_ip_host_field" decode="true"
        type="ip_address" minlen="4" maxlen="6" seqname="login_ip_host_sequence" atttypesize="1"
        attlengthsize="1" parentsize="0" />
- <!--
    LOGINSERVICE
    -->
- <protocolattribute protocolvalue="15" maptoname="login_service_field" decode="true"
        type="hashmap" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
        parentsize="0">
    - <hashmap>
        - <!--
            The mappings of the Login Service Attribute to the string value output in the RV message
            -->
            <hashmapelement id="0" value="Telnet" />
            <hashmapelement id="1" value="RLogin" />
            <hashmapelement id="2" value="TCP Clear" />
            <hashmapelement id="3" value="PortMaster (proprietary)" />
            <hashmapelement id="4" value="LAT" />
            <hashmapelement id="5" value="X25-PAD" />
            <hashmapelement id="6" value="X25-T3POS" />
            <hashmapelement id="8" value="TCP Clear Quiet (suppresses any NAS-generated
                connect string)" />
        </hashmap>
    </protocolattribute>
- <!--
    LOGINTCPPORT
    -->
    <protocolattribute protocolvalue="16" maptoname="login_tcp_port_field" decode="true" type="int"
        minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
    UNASSIGNED
    -->
    <protocolattribute protocolvalue="17" maptoname="NOTUSED" decode="false" type="string"
        minlen="0" maxlen="0" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
    REPLYMESSAGE
    -->
    <protocolattribute protocolvalue="18" maptoname="reply_message_field" decode="true" type="string"
        minlen="1" maxlen="253" seqname="reply_message_sequence" atttypesize="1"
        attlengthsize="1" parentsize="0" />
- <!--
    CALLBACKNUMBER
    -->
    <protocolattribute protocolvalue="19" maptoname="callback_number_field" decode="true"
        type="string" minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1"
        parentsize="0" />
- <!--
    CALLBACKID
    -->
    <protocolattribute protocolvalue="20" maptoname="callback_id_field" decode="true" type="string"
        minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
    UNASSIGNED
    -->
    <protocolattribute protocolvalue="21" maptoname="NOTUSED" decode="false" type="string"
        minlen="0" maxlen="0" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
    FRAMEDROUTE
    -->
```

```
<protocolattribute protocolvalue="22" maptoname="framed_route_field" decode="true" type="string"
    minlen="1" maxlen="253" seqname="framed_route_sequence" atttypesize="1" attlengthsize="1"
    parentsize="0" />
- <!--
FRAMEDIPXNETWORK
 -->
<protocolattribute protocolvalue="23" maptoname="framed_ipx_network_field" decode="true"
    type="string" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
    parentsize="0" />
- <!--
STATE
 -->
<protocolattribute protocolvalue="24" maptoname="state_field" decode="true" type="string"
    minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
CLASS
 -->
<protocolattribute protocolvalue="25" maptoname="class_field" decode="true" type="string"
    minlen="1" maxlen="253" seqname="class_sequence" atttypesize="1" attlengthsize="1"
    parentsize="0" />
- <!--
VENDORSPECIFIC
 -->
- <protocolattribute protocolvalue="26" maptoname="vendor_id_field" decode="true" type="int"
    minlen="0" maxlen="4096" seqname="none" atttypesize="1" attlengthsize="1" parentsize="4">
  - <vsadictionary>
      <subattribute protocolvalue="1" maptoname="netzwert_wlan_ordernumber_field"
          decode="true" type="int" maxlen="4" minlen="4"
          seqname="netzwert_wlan_order_sequence" atttypesize="1" attlengthsize="1"
          parentsize="0" />
      <subattribute protocolvalue="2" maptoname="netzwert_wlan_productcode_field"
          decode="true" type="string" minlen="0" maxlen="253"
          seqname="netzwert_wlan_product_sequence" atttypesize="1" attlengthsize="1"
          parentsize="0" />
      <subattribute protocolvalue="3" maptoname="netzwert_wlan_timepurchased_field"
          decode="true" type="int" minlen="4" maxlen="4"
          seqname="netzwert_wlan_time_sequence" atttypesize="1" attlengthsize="1"
          parentsize="0" />
    </vsadictionary>
</protocolattribute>
- <!--
SESSIONTIMEOUT
 -->
<protocolattribute protocolvalue="27" maptoname="session_timeout_field" decode="true" type="int"
    minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
IDLETIMEOUT
 -->
<protocolattribute protocolvalue="28" maptoname="idle_timeout_field" decode="true" type="int"
    minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
TERMINATIONACTION
 -->
- <protocolattribute protocolvalue="29" maptoname="termination_action_field" decode="true"
    type="hashmap" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
    parentsize="0">
  - <hashmap>
    - <!--
      The mappings of the Termination Action Attribute to the string value output in the RV
        message
      -->
      <hashmapelement id="0" value="Default" />
      <hashmapelement id="1" value="RADIUS-Request" />
    </hashmap>
</protocolattribute>
- <!--
CALLEDSTATIONID
 -->
<protocolattribute protocolvalue="30" maptoname="called_station_id_field" decode="true"
    type="string" minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1"
    parentsize="0" />
- <!--
```

```
CALLINGSTATIONID
 -->
 <protocolattribute protocolvalue="31" maptoname="calling_station_id_field" decode="true"
      type="string" minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
NASIDENTIFIER
 -->
 <protocolattribute protocolvalue="32" maptoname="nas_identifier_field" decode="true" type="string"
      minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
PROXYSTATE
 -->
 <protocolattribute protocolvalue="33" maptoname="proxy_state_field" decode="true" type="string"
      minlen="1" maxlen="253" seqname="proxy_state_sequence" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
LOGINLATSERVICE
 -->
 <protocolattribute protocolvalue="34" maptoname="login_lat_service_field" decode="true"
      type="string" minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
LOGINLATNODE
 -->
 <protocolattribute protocolvalue="35" maptoname="login_lat_node_field" decode="true" type="string"
      minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
LOGINLATGROUP This is a 32bit map, so probably wont decode directly as a string
 -->
 <protocolattribute protocolvalue="36" maptoname="login_lat_group_field" decode="true"
      type="string" minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
FRAMEDAPPLETALKLINK
 -->
 <protocolattribute protocolvalue="37" maptoname="framed_appletalk_link_field" decode="true"
      type="int" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
FRAMEDAPPLETALKNETWORK
 -->
 <protocolattribute protocolvalue="38" maptoname="framed_appletalk_network_field" decode="true"
      type="int" minlen="4" maxlen="4" seqname="framed_appletalk_network_sequence"
      atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
FRAMEDAPPLETALKZONE
 -->
 <protocolattribute protocolvalue="39" maptoname="framed_appletalk_zone_field" decode="true"
      type="string" minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0" />
- <!--
ACCTSTATUSTYPE
 -->
- <protocolattribute protocolvalue="40" maptoname="acct_status_type_field" decode="true"
      type="hashmap" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
      parentsize="0">
   - <hashmap>
      - <!--
        The mappings of the Login Service Attribute to the string value output in the RV message
        -->
        <hashmapelement id="1" value="Start" />
        <hashmapelement id="2" value="Stop" />
        <hashmapelement id="3" value="Interim-Update" />
        <hashmapelement id="7" value="Accounting-On" />
        <hashmapelement id="8" value="Accounting-Off" />
        <hashmapelement id="15" value="Reserved for Failed" />
     </hashmap>
 </protocolattribute>
- <!--
ACCTDELAYTIME
```

```
 -->
 <protocolattribute protocolvalue="41" maptoname="acct_delay_time_field" decode="true" type="int"
       minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
 ACCTINPUTOCTETS
 -->
 <protocolattribute protocolvalue="42" maptoname="acct_input_octets_field" decode="true" type="int"
       minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
 ACCTOUTPUTOCTETS
 -->
 <protocolattribute protocolvalue="43" maptoname="acct_output_octets_field" decode="true"
       type="int" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
       parentsize="0" />
- <!--
 ACCTSESSIONID
 -->
 <protocolattribute protocolvalue="44" maptoname="acct_session_id_field" decode="true"
       type="string" minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1"
       parentsize="0" />
- <!--
 ACCTAUTHENTIC
 -->
- <protocolattribute protocolvalue="45" maptoname="acct_authentic_field" decode="true"
       type="hashmap" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
       parentsize="0">
   - <hashmap>
       - <!--
        The mappings of the Acct-Authentic Value Attribute to the string value output in the RV
           message
        -->
        <hashmapelement id="1" value="RADIUS" />
        <hashmapelement id="2" value="Local" />
        <hashmapelement id="3" value="Remote" />
     </hashmap>
 </protocolattribute>
- <!--
 ACCTSESSIONTIME
 -->
 <protocolattribute protocolvalue="46" maptoname="acct_session_time_field" decode="true"
       type="int" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
       parentsize="0" />
- <!--
 ACCTINPUTPACKETS
 -->
 <protocolattribute protocolvalue="47" maptoname="acct_input_packets_field" decode="true"
       type="int" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
       parentsize="0" />
- <!--
 ACCTOUTPUTPACKETS
 -->
 <protocolattribute protocolvalue="48" maptoname="acct_output_packets_field" decode="true"
       type="int" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
       parentsize="0" />
- <!--
 ACCTTERMINATECAUSE
 -->
- <protocolattribute protocolvalue="49" maptoname="acct_terminate_cause_field" decode="true"
       type="hashmap" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
       parentsize="0">
   - <hashmap>
       - <!--
        The mappings of the Acct-Terminate-Cause Value Attribute to the string value output in the
           RV message
        -->
        <hashmapelement id="1" value="User Request" />
        <hashmapelement id="2" value="Lost Carrier" />
        <hashmapelement id="3" value="Lost Service" />
        <hashmapelement id="4" value="Idle Timeout" />
        <hashmapelement id="5" value="Session Timeout" />
        <hashmapelement id="6" value="Admin Reset" />
```

```
            <hashmapelement id="7" value="Admin Reboot" />
            <hashmapelement id="8" value="Port Error" />
            <hashmapelement id="9" value="NAS Error" />
            <hashmapelement id="10" value="NAS Request" />
            <hashmapelement id="11" value="NAS Reboot" />
            <hashmapelement id="12" value="Port Unneeded" />
            <hashmapelement id="13" value="Port Preempted" />
            <hashmapelement id="14" value="Port Suspended" />
            <hashmapelement id="15" value="Service Unavailable" />
            <hashmapelement id="16" value="Callback" />
            <hashmapelement id="17" value="User Error" />
            <hashmapelement id="18" value="Host Request" />
        </hashmap>
    </protocolattribute>
- <!--
    ACCTMULTISESSIONID
    -->
    <protocolattribute protocolvalue="50" maptoname="acct_multi_session_id_field" decode="true"
        type="string" minlen="1" maxlen="253" seqname="acct_multi_session_id_sequence"
        atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
    ACCTLINKCOUNT
    -->
    <protocolattribute protocolvalue="51" maptoname="acct_link_count_field" decode="true" type="int"
        minlen="4" maxlen="4" seqname="acct_link_count_sequence" atttypesize="1" attlengthsize="1"
        parentsize="0" />
- <!--
    CHAPCHALLENGE
    -->
    <protocolattribute protocolvalue="60" maptoname="chap_challenge_field" decode="false"
        type="string" minlen="16" maxlen="16" seqname="none" atttypesize="1" attlengthsize="1"
        parentsize="0" />
- <!--
    NASPORTTYPE
    -->
- <protocolattribute protocolvalue="61" maptoname="nas_port_type_field" decode="true"
        type="hashmap" minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1"
        parentsize="0">
    - <hashmap>
        - <!--
          The mappings of the NAS-Port-Type Value Attribute to the string value output in the RV
            message
          -->
            <hashmapelement id="0" value="Async" />
            <hashmapelement id="1" value="Sync" />
            <hashmapelement id="2" value="ISDN Sync" />
            <hashmapelement id="3" value="ISDN Async V.120" />
            <hashmapelement id="4" value="ISDN Async V.110" />
            <hashmapelement id="5" value="Virtual" />
            <hashmapelement id="6" value="PIAFS" />
            <hashmapelement id="7" value="HDLC Clear Channel" />
            <hashmapelement id="8" value="X.25" />
            <hashmapelement id="9" value="X.75" />
            <hashmapelement id="10" value="G.3 Fax" />
            <hashmapelement id="11" value="SDSL - Symmetric DSL" />
            <hashmapelement id="12" value="ADSL-CAP - Asymmetric DSL, Carrierless
                Amplitude Phase Modulation" />
            <hashmapelement id="13" value="ADSL-DMT - Asymmetric DSL, Discrete Multi-Tone"
                />
            <hashmapelement id="14" value="IDSL - ISDN Digital Subscriber Line" />
            <hashmapelement id="15" value="Ethernet" />
            <hashmapelement id="16" value="xDSL - Digital Subscriber Line of unknown type" />
            <hashmapelement id="17" value="Cable" />
            <hashmapelement id="18" value="Wireless - Other" />
            <hashmapelement id="19" value="Wireless - IEEE 802.11" />
        </hashmap>
    </protocolattribute>
- <!--
    PORTLIMIT
    -->
    <protocolattribute protocolvalue="62" maptoname="port_limit_field" decode="true" type="int"
        minlen="4" maxlen="4" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
- <!--
```

```
        LOGINLATPORT
        -->
        <protocolattribute protocolvalue="63" maptoname="login_lat_port_field" decode="true" type="string"
            minlen="1" maxlen="253" seqname="none" atttypesize="1" attlengthsize="1" parentsize="0" />
    </tlvattributedefs>
</TLVDecoderConfig>
```

Exceptions to the TLV Rule

The purity of the TLV approach to decoding the variable attribute section of a RADIUS message is complicated by CHAP PASSWORD and VENDOR SPECIFIC attributes. CHAP PASSWORD is carried as Type, Length, CHAP Identifier (a one octet value that represents the CHAP Identifier from the user's CHAP response and String (16 octets containing the CHAP response). As CHAP Identifier is essentially being carried as an embedded attribute, this throws off the purity of TLV for this attribute.
There are similar complexities with handling the 'Vendor Specific' attribute. The XML model recognises this, and models the VSA dictionary as a child attribute of vendor_id (as pictured in the fragment below).

```
- <protocolattribute protocolvalue="26" maptoname="vendor_id_field" decode="true" type="int"
      minlen="0" maxlen="4096" seqname="none" atttypesize="1" attlengthsize="1" parentsize="4">
    - <vsadictionary>
        <subattribute protocolvalue="1" maptoname="netzwert_wlan_ordernumber_field"
            decode="true" type="int" maxlen="4" minlen="4"
            seqname="netzwert_wlan_order_sequence" atttypesize="1" attlengthsize="1"
            parentsize="0" />
        <subattribute protocolvalue="2" maptoname="netzwert_wlan_productcode_field"
            decode="true" type="string" minlen="0" maxlen="253"
            seqname="netzwert_wlan_product_sequence" atttypesize="1" attlengthsize="1"
            parentsize="0" />
        <subattribute protocolvalue="3" maptoname="netzwert_wlan_timepurchased_field"
            decode="true" type="int" minlen="4" maxlen="4"
            seqname="netzwert_wlan_time_sequence" atttypesize="1" attlengthsize="1"
            parentsize="0" />
      </vsadictionary>
  </protocolattribute>
-   <!
```

Length Checks

When building a generic TLV decoder to decode RADIUS, it is important to model and accurately check boundaries and in particular attribute lengths. Such vulnerabilities have been previously discovered in RADIUS protocol decoder implementations, and these should not be overlooked in a bespoke implementation. For further reference please refer to [1].

References

[1] Rafail Jason, Vulnerability Note VU#936683, “Multiple implementations of the RADIUS protocol do not adequately validate the vendor-length of the vendor-specific attributes”, version 18, 04/16/2002. http://www.kb.cert.org/vuls/id/936683.

[2] Rigney C., Willens S., Rubens A., Simpson W., "RFC2865 - Remote Authentication Dial In User Service (RADIUS)", June 2000. http://www.ietf.org/rfc/rfc2865.txt?number=2865

[3] Rigney C., “RFC 2866 RADIUS Accounting”, June 2000 http://www.ietf.org/rfc/rfc2866.txt?number=2866

[4] Hill Joshua, “An Analysis of the RADIUS Authentication Protocol”, Nov 24, 2001. http://www.untruth.org/~josh/security/radius/radius-auth.html

[5] Funk Software White Paper - “The Role of RADIUS/AAA in 3G/Mobile IP-Based Environments. http://www.funk.com/radius/Solns/3g_wp.asp

[6] Funk Software White Paper – “The Role of RADIUS/AAA in GSM Wireless Data Networks GPRS, EDGE, UMTS” http://www.funk.com/radius/Solns/gsm_wp.asp

[7] VSA Dictionary Lists http://www.freeradius.org/radiusd/share/

[8] Rivest, R., “RFC1321 - The MD5 Message-Digest Algorithm”, April 1992. http://www.ietf.org/rfc/rfc1321.txt?number=1321

[9] Cisco Systems – Understanding RADIUS http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cnsar/1_7/referenc/radius.htm

[10] Heinz Johner, Larry Brown, Franz-Stefan Hinner, Wolfgang Reis, Johan Westman, Understanding LDAP (IBM Redbook) http://www.redbooks.ibm.com/redbooks/SG244986.html

[11] OMG Object Management Group – “Introduction to OMG's Unified Modeling Language™ (UML®)” http://www.omg.org/gettingstarted/what_is_uml.htm