Adding bandwidth specification to a AAA Sever

JIA ZHOU

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2008

COS/CCS 2008-19

Adding bandwidth specification to a AAA Sever

Jia Zhou

September 14th, 2008

Supervisor & Examiner: Professor Gerald Q. Maguire Jr.

Department of Communication Systems

School of Information and Communication Technology

Royal Institute of Technology

Stockholm, Sweden

Abstract

Authentication, authorization, and accounting (AAA) are key elements in network security. In many networks, clients can use resources only after they have been authenticated by an authentication server and authorized to use these resources. In some cases the server will also maintain accounting records in order for an operator (a provider of resources) to charge the account/subscriber for using the service. There are four main AAA protocols being used today. Of these RADIUS is the mostly widely used.

This thesis starts with an introduction to AAA protocols, and then goes in the details of RADIUS. In order to perform a practical evaluation of how the AAA could be improved, FreeRADIUS was selected as the base code for this project; because this implementation is one of the most widely used RADIUS servers. A proposal for how to improve AAA performance is introduced and the implementation steps needed to realize these improvements are shown. Additionally, some experiments have been conducted to show both the correct functioning of the resulting implementation and to examine if there is a performance improvement. Following this some conclusions are drawn based upon a comparison with a traditional AAA server.

A key element of the change in AAA which is proposed is the use of a non-binary IEEE 802.1x process. This new non-binary solution introduces a new type of AAA server and requires the re-thinking of a number of traditional AAA design decisions. It is expected that this change will have a significant impact, but will require some time for exposure, implementation by others, and a more extensive evaluation that was possible during the period of this thesis project.

One of the most important conclusions drawn during this thesis is the difficulty of making a change in authentication and authorization, because of the large amount of interaction between both the various protocols and the standards which have been developed for these protocols. Thus one of the difficult aspects of the task is how to introduce a change in a protocol while maintaining backward compatibility for others who have not adopted this change -- without requiring the addition of a protocol version field.

A second important conclusion is that doing this implementation in three separate parts with different students being responsible for the different parts revealed just how complex the interaction of protocol design decisions are. While a working version of the entire set of changes proved to be impossible, it was observed that the different parts could be decoupled more than initially expected.

Keywords: AAA, RADIUS, FreeRADIUS, authentication, non-binary authentication, IEEE 802.1x.

Acknowledgement

This thesis lasts for more than 6 months, and I dare say what I have learnt during these 6 months is the most important thing in my master studying life.

At first, I would like to show my sincere thanks to my Supervisor & Examiner: **Professor Gerald Q. Maguire Jr**. He is such a smart, erudite, insightful man leading me to a new period of study. During my thesis time, he showed his great patience to help me learn about this field, work with the RADIUS sever and Linux system, solve all the problems I have faced, and improve my English writing. He helped me understand more things about the AAA server and teach me what kind of characteristic a master should have. Learning, thinking, and the tireless efforts would be the spirit I will never drop in my life. His positive and friendly attitude rewarded me a great time in Wireless@KTH Lab.

I will also show my thanks to all my friends, especially to: Zhang Jiayi, Dai Kaiyu, Zuo Lin, Han Liu, Guo Jia, Zhang Hengchong, and Hu Lidan. Thanks for their patience when I had problems with the thesis. They discussed with me, worked together with me, and help me out with the problems blocked me. And thanks for their encouragement which makes me never give up.

Moreover, thanks go to all the staff at Wireless@KTH Lab. It is really wonderful to work together with them.

I will thank all my family members and best friends in China. Even they are not beside me, they give me courage and strength to work hard on this thesis.

Thanks to all the people!

Table of Contents

List of Figures

List of Table

List of Acronyms and Abbreviations

AAA	Authentication, Authorization, and Accounting
AVPs	Attribute-Value Pairs
CHAP	Challenge-Handshake Authentication Protocol
CMS	Cryptographic Message Syntax
DSL/xDSL	Digital subscriber line
EAP	Extensible Authentication Protocol
IAS	Internet Authentication Service
IETF	Internet Engineering Task Force
IPsec	Internet Protocol Security
ISP	Internet Service Provider
LDAP	Lightweight Directory Access Protocol
LEAP	Localized Encryption and Authentication Protocol
MD5	Message Digest 5
MIP	Mobile IP
MN	Mobile Node
NAS	Network Access Server
OS	Operating System
PAP	Password Authentication Protocol

PEAP	Protected Extensible Authentication Protocol
PPP	Point-to-Point Protocol
RADIUS	Remote Authentication Dial In User Service
SQL	Structured Query Language
TACACS	Terminal Access Controller Access-Control System
TACACS+	Terminal Access Controller Access-Control System Plus
TLS	Transport Layer Security
UDP	User Datagram Protocol
VOIP	Voice over IP
VPN	Virtual private network
WLAN	Wireless Local Area Network

Chapter 1: Introduction to AAA

Authentication, authorization, and accounting (AAA) [1] are part of a network's security. They provide the framework for access control as enforced by a router or access server. [2] While one might think that all network resources can be managed using an authentication, authorization, and accounting system; this is not strictly true. There are additional network management functions and functions concerning, operations, maintenance, and provisioning - which we will ignore in this thesis. For commercial systems, authentication is considered a crucial issue – as the operator only wants to provide resources to legitimate subscribers. In this model, only when a subscriber has been identified (based upon authentication) can the system determine what services (if any) this subscriber is to be provided with and who will pay for the usage of this service. The system must also prevent attacks by unauthorized users, who may attempt to utilize a service for which they are not authorized. [3]

This thesis began with an observation by Gerald Q. Maguire Jr. that the traditional networking approach of doing authentication and authorization before providing any server (for example in the case of IEEE 802.1X port based access control - discussed in **section 2.2.2**): (1) comes at the price of high delay before the user can receive service and (2) many business based transactions are not based upon authentication and authorization before a service is provided as a large percentage of users will in fact pay for the services they use later - while only a small fraction will cheat. The first of these observations can be directly correlated with the problems in packet loss, loss of connectivity, etc. which moving nodes have today in many WLAN mobility settings. While the second observation is based upon the risk of management approach which credit card companies, restaurants, etc. take to ensure that the rate of fraud is low -- but do not employ means which have greater cost than the potential loss due to a user not paying! This thesis is one of three thesis projects taking place to examine what would happen if users were enabled to have some service before they were authenticated and authorized. Details of the related theses can be found in Guo Jia's thesis [4] and Zhang Hengchong's thesis [5].

This thesis will focus on the authentication and authorization server, explicitly the case of a RADIUS server, and its interaction with an alternative to an IEEE 802.1x authenticator being developed by Guo Jia.

1.1 AAA Protocol

As AAA concerns three elements, we will introduce these three parts along with the protocols used [1] [2] [3]:

Authentication refers to the process of establishing the digital identity of an entity to another entity (such as a client computer to a server computer). In a network, authentication is configured by defining a named list (or unnamed default list) of authentication methods, then applying that list of methods to various interfaces. To validate the subscriber’s identity, the system considers the claimed identity and its corresponding credentials. Passwords, digital certificates, one-time tokens, and even physical/biological characteristics may be utilized as credentials.

Authorization refers to the granting of specific types of privileges (including "no privilege") to an entity, based on their authentication, what privileges they are requesting, and the current system state. Authorization may include special restrictions, such as time restrictions, physical location restrictions, or restrictions against multiple logins by the same user. Granting access to the service defines the set of services a subscriber may utilize; i.e. their actual privileges and restrictions. Once the subscriber has been authorized to use the service, they can now proceed to actually use the service – often without any further checks - unless there is a metered limited to their usage. Note that here we have referred to a subscriber as the entity who has been authenticated and who is allowed to access a service. This definition of a subscriber should be considered broadly, in order to encompass post-pay subscribers, pre-paid subscribers, and even subscribers - who are allowed to access these services because someone else is paying for these services.

Accounting refers to the tracking the consumption of resources by subscribers. This function collects information such as subscriber identities, start and stops times, and executed commands. All of this information may be used for management, planning, billing, or other purposes. Real-time accounting refers to accounting information that is delivered concurrently with the consumption of the resources. Batch accounting refers to accounting information that is saved until it is processed at a later time. Generally the index for all accounting records is the subscriber’s identity (or an indirect representation of it – such as an account number).

These three functions enable the network to record who is using the network resources and to determine whom should be allowed to use these resources. Additionally, these three elements help the operator (actually, the service provider) to ensure that the subscriber is provided with the services to which they have subscribed; while helping to avoid providing service to those who have not subscribed for the service.

1.2 AAA Protocol application

In a traditional telecommunication system, users can only use network resource after establishing that they should be allowed to use this resource. Generally, the authentication and authorization processes involve three entities: Client, Authenticator, and AAA Server. In mobile communication systems, the client is often called a Mobile Node (MN). [3]

Consider the case of dial up remote access. In this case, as shown in **Figure 1.1**, an authenticator is located at a Network Access Server (NAS). Because in this figure we consider the case of dialup access, the authenticator generally uses the Point-to-Point Protocol (PPP) [6] to communicate with the client. The AAA protocol used between the Authenticator and the AAA Server is one of several AAA protocols (which we will examine in the next section).

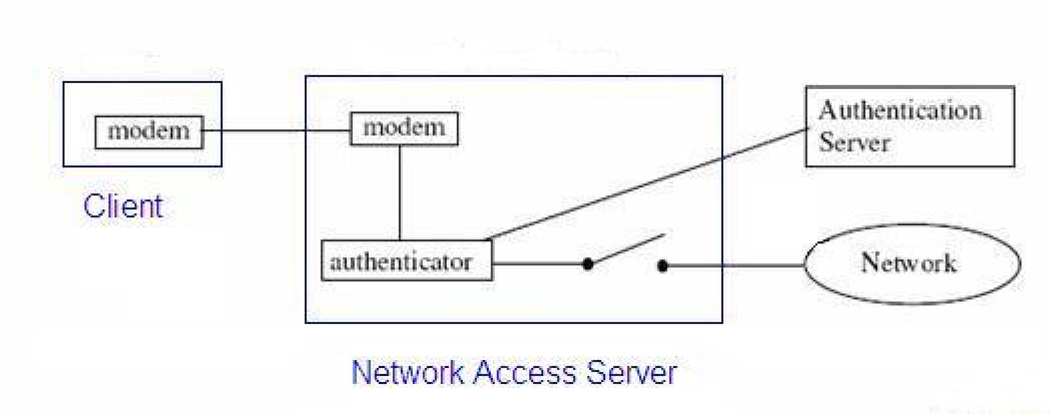


Figure 1.1 An example of AAA in the case of dial-up network access

In the case of dialup access, when clients try to connect to the network, they need to be authenticated. Once they have been authenticated and authorized, then the switch will be closed and they will have access to the network. The Authentication Server controls this switch. After the session ends or at some point in time (when the authorized usage ends), then the switch will be opened and access to the network will be denied. Note that while we refer to the control of the access to the service as being a switch, we will see that in traditional authorization systems it is a binary control (i.e., either access to the resource is permitted or it is not) and in the proposed new solution we will see that rather than a binary switch, we can use a traffic shaper or other device to provide **limited** access to a resource - where these limitations can be much more varied than simply on or off.

1.3 AAA Protocols

There are four main AAA protocols currently being used: RADIUS, Diameter, TACACS, and TACACS+. Sometimes these protocols are used in combination with Point-to-Point Protocol (PPP), Extensible Authentication Protocol (EAP), Protected Extensible Authentication Protocol (PEAP), and Lightweight Directory Access Protocol (LDAP). The following subsections will introduce each of these protocols.

1.3.1 RADIUS

Remote Authentication Dial In User Service (RADIUS) is commonly used by ISPs

(Internet Service Providers) and corporations for access control. It is primarily used to manage access to the internet or other networks. These networks can employ a variety of networking technologies, including analog modems, DSL, wireless local area networks (WLANs), and VPNs. [7]

RADIUS is based on the UDP (User Datagram Protocol). As shown earlier in **Figure 1.1**, in the case of a dialup services, the NAS server acted as a RADIUS client when it contacted the authentication server. This was the original use of RADIUS and the origin of its name. However, today any computer which runs RADIUS Client software can be a RADIUS client. The authentication mechanism of RADIUS is quite flexible and offers a variety of ways to authenticate the user (in the case of some dial up access this is referred to as logging in), such as: PAP, CHAP, or UNIX. [8]

RADIUS carries parameters using a vector of Attribute-Length-Value (often referred to as S server will check the validity of the user name and password. It can also referred as Attribute-Value Pairs (AVPs)) entries. RADIUS also allows manufacturers to extend it by adding their own attributes. [8]

1.3.1.1 Basic operation of RADIUS

The basic operation of RADIUS is as follows:

a) User connects to the NAS; the NAS sends an Access-Require packet to the RADIUS service. This packet contains user information, such as: user name and password. The password will be hashed using MD5; therefore, both sides must know this password which acts as a shared secret key. Note that the plain text of this key will not be transmitted over the network. (Note that techniques such as described in RFC 2085 can be employed to perform a keyed MD5 hash with replay prevention. [9] However, in the simplest case if the service provider assumes that the client is connected to the NAS via a physically secure connection - therefore MD5 alone might be used.)

b) The RADIUS server will check the validity of the user name and password. It can also return a challenge that can be used to authenticate either the user or the NAS.

c) If the authentication is successful, then the RADIUS server will send an **`Access-Accept`** packet to the NAS - thus allowing the user access to the network; otherwise, it will return an **`Access-Reject`** packet and the NAS will refuse the user access.

d) Once the user is allowed access, the NAS will send a charging requirement (**`Account-Require)`** to the RADIUS service. The RADIUS server answers with an **`Account-Accept`** message, the subscriber’s account will now begin to accrue a fee for the service. Periodically, Interim Accounting records may be sent by the NAS to the RADIUS server, to update it on the status of an active session.

e) Finally, when the user's network access ends, the NAS issues a final **`Accounting-Stop`** message to the RADIUS server, providing information on the final usage in terms of time, packets transferred, data transferred, and the reason for disconnect, and other information related to the user's network access. Therefore, the server will not continue to charge the subscriber any longer.

There are also some other functions in RADIUS, such as: proxy operation service, roaming service, repeating mechanism, etc. [8] However, for the purpose of this thesis understanding the above basic operations is sufficient.

1.3.1.2 RADIUS message exchange flow

As described about, the authenticator (access point/RADIUS client) authenticates users via the RADIUS server. The details of this as shown below: [10]

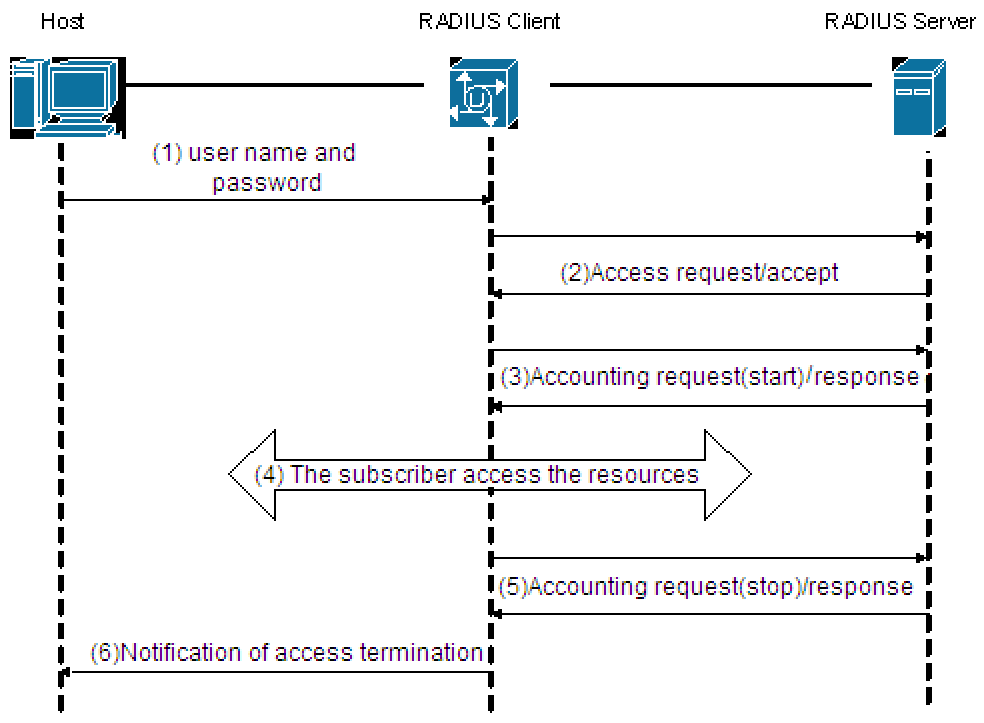

Host
RADIUS Client
RADIUS Server
(1) user name and password
(2)Access request/accept
(3)Accounting request(start)/response
(4) The subscriber access the resources
(5)Accounting request(stop)/response
(6)Notification of access termination

Figure 1.2 Basic message exchange process of RADIUS

1.3.2 Diameter

Diameter is a successor to RADIUS. It is not directly backwards compatible, but rather provides an upgrade path for RADIUS. [11] The Diameter protocol contains a base protocol, a NAS protocol, an EAP protocol, a MIP protocol, a CMS protocol, etc. Each of these will be described below: [3]

- ✧ The base protocol of Diameter provides the elementary service for Mobile IP, NAS, and so on. The base protocol allows the transfer of commands and AVPs, and it can transit AAA information between clients, proxy, and server.
- ✧ The NAS protocol (of Diameter) is simply the Network Access Service protocol.
- ✧ Diameter EAP, which stands for Extensible Authentication Protocol, provides a standard mechanism to support all kinds of authentication (hence the word “extensible”).
- ✧ Diameter MIP (Mobile IP) allows user roaming to exterior regions; the user can use the service provided by an exterior region server or agent after getting authorization.
- ✧ Diameter CMS (Cryptographic Message Syntax) protocol provides Peer-to-Peer encryption to protect the protocol data.

Diameter Applications can extend the base protocol, by adding new commands and/or attributes. An application is not a program, but rather a ***protocol*** based on Diameter. Diameter security is provided by IPsec [12] or TLS [13], both well-regarded security and privacy protocols. [11]

1.3.3 TACACS

Terminal Access Controller Access-Control System (TACACS) is a remote authentication protocol that is used to communicate with an authentication server. It is commonly used in networks of UNIX systems. TACACS allows a remote access server to communicate with an authentication server in order to determine if the user should have access to the network. [14]

Each TACACS client has a user name and password. The client can send a query to a TACACS authentication server (sometimes called a TACACS daemon or simply TACACSD). This server is normally a program running on a host. The program determines whether to accept or deny the request and sends a response back. The TIP (a node accepting dial-up connection would then allow access or not, based upon the response. In this approach the decision making process is "opened up" and the algorithms and data used to make the decision are completely under the control of whoever is running the TACACS daemon. [14]

1.3.4 TACACS+

TACACS+ (Terminal Access Controller Access-Control System Plus) is a protocol which provides access control for routers, network access servers, and other networked computing devices via one or more centralized servers. [15] TACACS+ is based on TACACS, but, in spite of its name, it is an entirely new protocol which is incompatible with all previous versions of TACACS. TACACS+ and RADIUS have generally replaced the earlier access control protocols in more recently built or updated networks, although TACACS and XTACACS are still running on many older systems. [15] TACACS+ utilizes TCP port 59. The operation of TACACS+ is shown **Figure 1.3**. [16] The sequence of the messages which are sent is shown in the figure. The primary importance of this numbering is simply to shown that just as in the case of RADIUS and Diameter, the NAS queries the server to determine if the user should be allowed access to a resource; i.e., the NAS does not **make** the decision, but does implement the result of the decision made by the server.

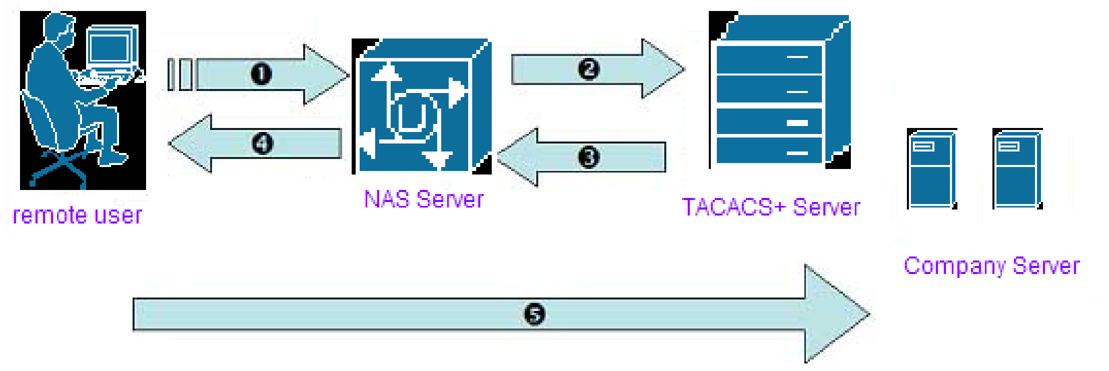


Figure 1.3 TACACS+ overview

If the client wants use a resource, it needs to authenticate and get authorization from the TACACS+ server. TACACS+ offers multiprotocol support (supporting both IP and AppleTalk). Normally TACACS+ fully encrypts the body of the packet to provide secure communications. TACACS+ is a Cisco proprietary enhancement to the original TACACS protocol. [15]

1.4 Summary

In this research project, RADIUS was chosen as it is one of most commonly used AAA protocols and multiple open source implementations exist (one of which will be used as the basis for this thesis project). In the next chapter we will take a deeper look at RADIUS.

Chapter 2: Introduction to RADIUS

2.1 Background

The RADIUS protocol was originally developed by the Livingston Company for authenticating and charging of dial-up users. RADIUS was adopted in 1991 by Merit Network, Inc. (a non-profit company belonging to the University of Michigan) for the MichNET university network. This leads to wide adoption by many others. [8]

In the autumn of 1992, IETF founded a work group named NASREQ that worked with a draft proposal for RADIUS. RADIUS became an internet access protocol soon after that. Almost all internet access server providers have implemented this protocol. [8] Since then RFC2039 [17], RFC2138 [18], RFC2865 [19], and RFC2866 [20] have extended the definition of RADIUS. RADIUS remains in wide use, despite the specification and development of DAIMETER.

2.2 RADIUS details

A RADIUS service involves three components: Protocol, Server, and Client. In the client/server model of RADIUS, the client, such as a router or a switch, passes user information to the designated RADIUS server and acts on the response of the server (such as connecting/disconnecting users). The RADIUS server receives authentication requests from a RADIUS client (authenticator), authenticates users, and returns the required information to the client. [10]

In general, a RADIUS server maintains three databases: [10]

Subscribers	This database stores user information such as the username, password, and IP address.
Clients	This database stores information about RADIUS clients such as the shared key for each client.
Dictionary	This database stores the information necessary for interpreting RADIUS protocol attributes and their values.

RADIUS has one basic message exchange and a single packet structure. The basic message exchange process has been introduced in **section 1.3.1**. Here we will introduce the packet structure, and IEEE 802.1x authenticator is also introduced in following subsections.

2.2.1 RADIUS packet structure

RADIUS resides at the application layer in the TCP/IP protocol suite. In conjunction with the RADIUS protocol it defines how to exchange information between a RADIUS client and a RADIUS server. [10]

RADIUS uses UDP to transport its messages. It uses UDP port 1812 for RADIUS authentication messages and UDP port 1813 for RADIUS accounting messages.[1] Exactly one RADIUS packet is encapsulated in a UDP payload. [21]

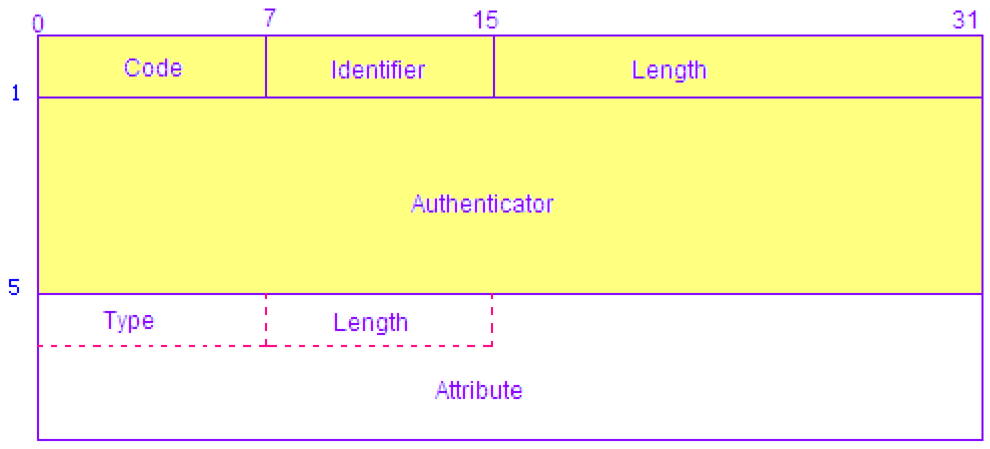


Figure 2.1 RADIUS packet structure

2.2.1.1 Code

The Code field is 1 byte in length and indicates the type of the RADIUS packet. A packet with an invalid Code field is silently discarded. **Table 2-1** introduces a number of different code values which are most commonly used. [10][21]

[1] Some older network access servers use UDP port 1645 for RADIUS authentication messages and UDP port 1646 for RADIUS accounting messages. Microsoft's Internet Authentication Service (IAS) (their implementation of RADIUS) supports the receiving of RADIUS messages on both sets of UDP ports.

Table 2-1: Values for most used RADIUS Code field

Code	Packet type	Sender	Description
1	Access-Request	Client	Sent from the client to the server. A packet of this type carries user information for the server to authenticate the user. It must contain the User-Name attribute.
2	Access-Accept	Server	Sent from the server to the client. A message with this code is sent when authentication succeeds.
3	Access-Reject	Server	Sent from the server to the client. If any attribute value carried in the Access-Request is unacceptable, the request will be rejected.
4	Accounting-Request	Client	Sent from the client to the server. A packet of this type carries user information for the server to start accounting. It contains the Acct-Status-Type attribute, which indicates whether the server is requested to start the accounting or to end the accounting.
5	Accounting-Response	Server	From the server to the client. The packet of this type is to notify that it has received the Accounting-Request and has correctly recorded the accounting information.
11	Access-Challenge	Server	If the RADIUS server desires to send the user a challenge requiring a response, then the RADIUS server MUST respond to the Access-Request by transmitting a packet with this message code.

2.2.1.2 Identifier

The identifier field is 1 byte in length and it is used to match a request with its corresponding response. The value in the reply is equal to the value in request. [22] It varies with the attribute field and the received valid response packets, but remains unchanged during retransmission. [10]

2.2.1.3 Length

The Length field is two bytes in length. This field indicates the length of the entire packet, including the Code, Identifier, Length, Authenticator, and Attribute fields. Bytes in the UDP payload beyond this length are considered padding and are neglected at receipt. The length field can vary from 20 to 4,096, indicating the number of bytes in the entire packet bytes. If the length of a received packet is less than that indicated by the Length field, the packet is dropped. [10][21]

2.2.1.4 Authenticator

The Authenticator field is sixteen bytes in length and contains the information that the RADIUS client and server use to authenticate each other. There are two kinds of authenticators: Request and Response. [10]

The authenticator value in a request is randomly generated. While the value in the reply is MD5 digest of a reply message appended with the secret, using a vector value from the request. [22]

2.2.1.5 Attributes

The Attributes section of the RADIUS packet contains one or more RADIUS attributes, which carry the specific authentication, authorization, information, and configuration details for RADIUS. For attributes that have multiple instances, the order of the attributes must be preserved. Otherwise, attribute types do not have to have their order preserved.

This field contains triplets of Type, Length, and Value (as shown in **Figure 2.1**). [10]

- Type: This is only one byte in length and indicates the type of the attribute. **Table 2-2** lists the most commonly used attributes.
- Length: This field is one byte in length and indicates the length of the attribute in bytes, including the Type, Length, and Value fields.
- Value: Value of the attribute, up to 253 bytes. Its format and content depend on the Type and Length fields.

Table 2-2: Commonly used RADIUS attributes

Type	Attribute type	Type	Attribute type
1	User-Name	2	User-Password
3	CHAP-Password	4	NAS-IP-Address
5	NAS-Port	6	Service-Type
7	Framed-Protocol	8	Framed-IP-Address
9	Framed-IP-Netmask	10	Framed-Routing
11	Filter-ID	12	Framed-MTU
13	Framed-Compression	14	Login-IP-Host
15	Login-Service	16	Login-TCP-Port
17	(unassigned)	18	Reply-Message
19	Callback-Number	20	Callback-ID
21	(unassigned)	22	Framed-Route
23	Framed-IPX-Network	24	State
25	Class	26	Vendor-Specific
27	Session-Timeout	28	Idle-Timeout
29	Termination-Action	30	Called-Station-Id
31	Calling-Station-Id	32	NAS-Identifier
33	Proxy-State	34	Login-LAT-Service
35	Login-LAT-Node	36	Login-LAT-Group
37	Framed-Apple Talk-Link	38	Framed-Apple Talk-Network
39	Framed-Apple Talk-Zone	40-59	(reserved for accounting)
60	CHAP-Challenge	61	NAS-Port-Type
62	Port-Limit	63	Login-LAT-Port

2.2.2 IEEE 802.1x authenticators

What is IEEE 802.1x? As described in the IEEE 802.1x-2004 standard:

“*Port-based Network Access Control makes use of the physical access characteristics of IEEE 802 LAN infrastructures in order to provide a means of authenticating and authorizing devices attached to a LAN port that has point-to-point connection characteristics, and of preventing access to that port in cases in which the authentication and authorization process fails. A port in this context is a single point of attachment to the LAN infrastructure.*”[23]

IEEE 802.1X is a part of the IEEE 802.1 group of networking protocols. It provides a port-based Network Access Control and is used to provide compatible authentication and authorization mechanisms for devices interconnected by IEEE 802 LANs. [24] Although RADIUS support is optional within IEEE 802.1X, it is expected that many IEEE 802.1X Authenticators will function as RADIUS clients. [25] The IEEE 802.1x uses Extensible Authentication Protocol (EAP) to support many authentication methods - usually EAP-TLS. In **Figure 2.2**, the stack of 802.1x protocol is listed. [26] (Further details RADIUS usage by IEEE 802.1x Authenticators can be found in reference RFC 3580. [25])

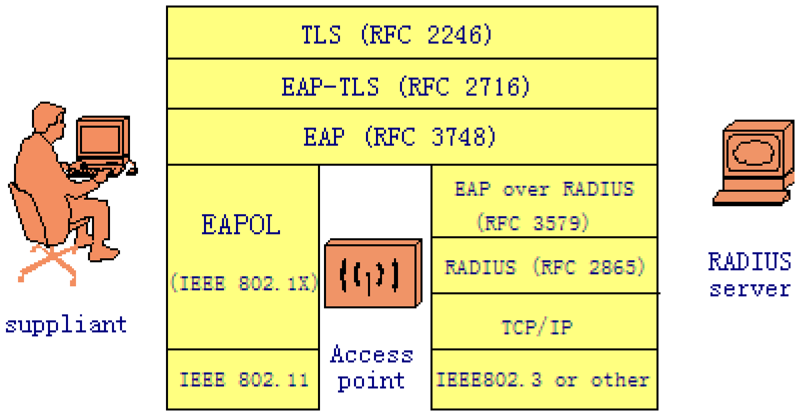


Figure 2.2 IEEE 802.1x protocol stack

2.3 Widely used RADIUS servers

As RADIUS is very commonly used today, there are many different implementations of RADIUS servers. Some are open source, which offers access to the source code; [18] while the others are proprietary (and generally designed for commercial use).

2.3.1 Open source

Open source software projects are built and maintained by a network of volunteer programmers. [27] One benefit of open source is that anyone can easily get the original code and improve it. While there is some criticism that this means that everyone has a chance to get the original code and attack it; history has shown that open source code also enables problems to be fixed quickly and that security holes are more likely to be found (see for example the number of bugs reported and addressed in an open source based router as compared to non-open source routers in Sara Dannerud's thesis [28]).

There are a number of open source RAIDUS servers used today, with most of them being viewed as secure enough to satisfy almost all customers’ requirements.

2.3.1.1 FreeRADIUS

FreeRADIUS was developed by the FreeRADIUS Development Team. It is one of the most modular and feature-rich RADIUS servers available today. The newest Version 2.0.5 was released in June 7th 2008. [29]

Additionally, FreeRADIUS scales from embedded systems with small amounts of memory, to systems with multiple millions of users. The FreeRADIUS server comes with a PHP based web user administration tool, called “dialupadmin”. This code base is currently used as the foundation for multiple commercial RADIUS products. [29]

FreeRADIUS supplies the AAA needs of many Fortune-500 companies and Tier 1 ISPs [30]. In addition to being widely used in commercial settings, it is also widely used by the academic community.

2.3.1.2 GNU Radius

GNU Radius is a centralized user authentication and accounting system. It supports back-end SQL databases for accounting. [31] However, GNU Radius had some vulnerabilities which needed to be addressed. An example of one such vulnerability has been insufficient filtering of user entries, which might permit a SQL injection attack (i.e., an entry which contains SQL attack code in it) -- this specific vulnerability was fixed in **version 1.4**.

2.3.1.3 OpenRADIUS

OpenRADIUS is a RADIUS server that runs on many variations of UNIX, and has a number of interesting features. Some of these features are its versatile interface to the outside world, flexibility for controlling processes, and ability to define user profiles. It also has a powerful dictionary which can support all types of vendor-specific attributes. [32]

2.3.1.4 Cistron RADIUS

Cistron RADIUS is an authentication and accounting server for terminal servers. It was the parent of the FreeRADIUS project. [33] The current version of it 1.6.8, which is last updated in February 8th 2006. [34]

2.3.1.5 BSDRadius

BSDRadius is an open source RADIUS server targeted for use in Voice over IP (VoIP) applications. Typically a VoIP RADIUS server should be able to process a large number of AAA requests in a short time period, handle large databases, and respond in a timely fashion to prevent time-outs and request retransmissions. [35] The features of BSDRadius are shown in **Table 2-3**.

BSDRadius uses a powerful library, pyrad, for lower level operations, such as parsing attribute dictionaries and building accounting and authorization packets. Due to its extensive use of Python, BSDRadius is as portable as Python is. Therefore it should run on any distribution of Linux or any flavor of BSD (FreeBSD, OpenBSD and NetBSD). [35]

Table 2-3: BSDRadius features

RADIUS - compliant AAA (Authentication, Authorization, Accounting) server
Various database engines: MySQL, PostgreSQL (support for Oracle is under development)
CHAP-password authentication for H.323
Digest authentication for SIP
Vendor specific dictionary files (Cisco, Quintum).
Very easy to use custom module interface
Storing RADIUS server client data into database.
Platform independent
Logging of received, failed and rejected requests to separate files for easier later processing
Comfortable framework for building RADIUS applications
Ready to use CLI RADIUS client

2.3.1.6 TekRADIUS

TekRADIUS is a free RADIUS server for Windows. TekRADIUS complies with RFC 2865 [19] and RFC 2866 [20]. It currently supports only Microsoft's SQL Server. It runs as a Windows Service and comes with a Win32 management interface. [36]

Windows users in the "Administrators" group can access all functions of the TekRADIUS Manager GUI. However, Windows users in the built-in "Users" group can only access a restricted set of functions in the TekRADIUS Manager GUI. [36]

2.3.2 Commercial products (appliances)

As the RADIUS servers are at the heart of the RADIUS authentication fabric, they need to be trusted implicitly by all parties and stakeholders. Running RADIUS on an appliance rather than a general-purpose computer is thought to provide a much better starting point to achieve this trust. [37] Therefore, there are several RADIUS appliances which act as RADIUS servers. These are primarily designed for commercial use.

2.3.2.1 Infoblox RADIUSone

Infoblox Inc. sells a RADIUS appliance under the product name: RADIUSone. Running the RADIUS service on its own platform avoids unexpected and unwanted interactions with other services. Hence this device only runs the software necessary for this specific service and no other software. RADIUSone offer a built-in high availability capability configuration, thus two RADIUSone appliances can be wired together on a private connection; if the primary unit fails, the secondary, which has a mirrored configuration, will take over. [37][38]

2.3.2.2 Identity Engines Ignition

The Identity Engines™ Ignition™ appliance enables enterprises to implement centralized security policies despite the increasing complexity brought on by replicating identity stores, complex network design, and constantly changing organizational structures. [39] It supports RADIUS and allows the user of multiple user identity stores, specifically Microsoft Active Directory 2000 and 2003, Sun Java System Directory Server 5, Novell eDirectory 8.7, and Embedded User Store. [40]

2.3.2.3 Palette Mobilette Access Management Gateway

Palette's Mobilette ™ Solution is based upon intelligent rules. It is used to implement a RADIUS based client-server system. Palette's Access Management Gateway is a part of the Mobilette™ solution. [41]

There are additional commercial appliances, which will not be introduced here, but many are available in the market. The above was not meant to be a recommendation of any specific vendor(s), simply to illustrate the variety of commercial appliances which exist.

2.3.3 Commercial (software) products

There are many commercial AAA software packages, such as: Microsoft's Internet Authentication Service which is included with server editions of Microsoft's Windows; Alepo's Radius Server; Steel-Belted Radius; IEA Software RadiusNT/X; Radiator radius, etc. [42] However, as these are commercial products which are not open source, we will not consider them further - as part of this thesis project requires that we extend the information which is communicated between the RADIUS client and the server.

2.4 Vulnerability of RADIUS

Although RADIUS is now widely used and well developed, it still has some vulnerabilities. These vulnerabilities are either caused by the protocol or caused by a poor client implementation and exacerbated by the protocol. **Table 2-4** shows the main security problems. [43]

Table 2-4: RADIUS protocol vulnerabilities

The User-Password protection technique is flawed in many ways. It should not use a stream cipher, and it should not use MD5 as a cipher primitive.
The Access-Request packet is **not** authenticated at all.
Many client implementations do not create Request Authenticators that are sufficiently random.
The Response Authenticator is a good idea, but it is poorly implemented.
Many administrators choose RADIUS shared secrets with insufficient information entropy. Many client and host implementations artificially limit the shared secret key space.

Some of these vulnerabilities have been addressed with the continuing developing of the RADIUS protocol. However, there still remain weak points in the RADIUS protocol. Additionally, various RADIUS software implementations and appliances have their own special vulnerabilities. For example, authentication can be bypassed when radius-authentication is used in OpenBSD, this will allow unauthorized access.

2.5 Comparing RADIUS with Diameter

Diameter was defined to be an upgrade protocol of RADIUS. It has solved some known RADIUS problems and made some improvements. [44] Here a comparison was made between RADIUS and Diameter; we can see clearly which part that Diameter has improved.

✧ **Strict limitation of attribute data**

The reserved length for the data field of RADIUS in its attribute header is only one byte, which allows a maximum of 255 bytes.

Diameter reserves two bytes for this (thus allowing a maximum length of 16535 bytes).

✧ **Inefficient retransmission algorithm**

There is only one byte as 6 identifier field to identify retransmissions in RADIUS. This means the number of requests that can be pending is limited to a maximum of 255 requests.

Diameter has reserved four bytes which leads to maximum of 2^{32} pending requests. This avoids the limitation in RADIUS.

✧ **Inability to control flow to servers**

RADIUS operates over UDP. There is no standard scheme for UDP to regulate the flow.

Diameter has a scheme which regulates the flow of UDP packets (windowing scheme).

✧ **End-to-end message acknowledgement**

The Radius client expects a response after a request, in order to know if the request was successful or not, but the client does not know whether the request has been received by the server. Thus the client does not know if the request was received but it is taking the server some time to process the request or if the request was lost.

The Diameter client expects a failed response or an acknowledgement of the received request by the server.

✧ **Silent discarding of packets**

The RADIUS server will silently discard packets which do not contain the expected information, or which have errors. However, this might cause the client to think that the server is down as no response has been received. The client would then try to send request to a secondary server.

The Diameter server will send an error message back to the client to indicate the problem, rather than simply dropping the request.

✧ **No fail-over server support**

There is no way to indicate that a given RADIUS server is going down.

The Diameter supports Keep-alive messages to indicate that a server is going down for a time period.

✧ **Authentication replay attacks**

Any RADIUS client can generate a challenge response sequence when using PPP CHAP. These sequences can be intercepted by any RADIUS client or proxy server in the chain. This challenge response sequence can then be replayed by another RADIUS client at any time. (This problem was partly solved by the radius extension using the EAP protocol.)

In Diameter, those challenge/response attributes can be secured by using end-to-end encryption and authentication.

✧ **Hop-by-hop security**

RADIUS only supports hop-by-hop security, which means that it is easy to modify information at every hop and information can not be traced to its origin.

Diameter supports end-to-end security which guarantees that the information can not be modified without being noticed.

✧ **No support for user-specific commands**

RADIUS only supports vendor specific **attributes**, i.e., vendor specific **commands** are not supported.

The Diameter has support for vendor specific **commands** codes.

✧ **Heavy processing costs**

The RADIUS protocol does not impose any alignment requirements, which adds an unnecessary burden on most processors.

The Diameter protocol has a 32 bit alignment requirement, enabling messages to be handled more efficient by most processors.

This thesis will utilize FreeRADIUS to improve (extend) the RADIUS server. In the third chapter, details of this extension will be introduced.

Chapter 3: Improving AAA utilizing FreeRADIUS

3.1 Why select FreeRADIUS?

FreeRADIUS was selected in this research project for the following reasons:

- ✓ It is open source server; hence the source code can be utilized without financial cost.
- ✓ It is the most widely deployed RADIUS server in the world. Multiple commercial offerings used it as their base. Thus the potential impact of extending it is greater than applying these extension to other versions of RADIUS
- ✓ The FreeRADIUS development team has continued to develop it, so it is very widely used and seems to be stable in everyday.
- ✓ Many Fortune-500 companies use it for their AAA needs. Thus these same companies could use the extended version, hence fostering the spread of the advantages of non-binary access control.

By basing our improvements to AAA on this AAA protocol a lot of people can potentially benefit from these improvements. This is particularly important for the overall project, as non-binary authentication will only be adopted if it is both feasible and easy.

3.2 Theoretic improvements

This paper will extend the for FreeRADIUS server support two new features: (1) to support non-binary authentication and (2) to allow on-demand negotiation. Before we talk about this new solution, we first begin with a theoretical assessment of FreeRADIUS.

3.2.1 Analysis of the existing FreeRADIUS

As the most commonly used free software RADIUS server, the FreeRADIUS has a number of features that are the same as the other implementations. Additionally, it has some other features which are not found in any RADIUS implementation. [45] In the next paragraph, we list some of the features of FreeRADIUS before further analysis.

3.2.1.1 Major features of FreeRADIUS

3.2.1.1.1 Cross-platform issues and source code

The FreeRADIUS server can run on a number of platforms, including linux (all versions), FreeBSD, NetBSD, Solaris, and MAC OSX. Additional platforms such as: HP/UX, AIX, MINGW32, CygWin (Unix- style environment under Window NT.), and SFU (or Interix, for Windows XP) are supported by the current server, but according to the FreeRADIUS website are not fully tested. [45]

FreeRADIUS has been designed (and verified, by others) to work on a large number of processor and operation system architectures. This causes the installation of the server to become rather complex as it is necessary to specify which configuration is being targeted. However, a lot of the platforms have their own pre-built FreeRADIUS package, thus installation is very easy (generally no more difficult that installing any other pre-build software for these same platforms). While this enables the software to be easily installed, there are still many configuration parameters (“attributes”) which need to be set before the software can be used. These are addressed next. [45]

3.2.1.1.2 Additional Server configuration attributes

The FreeRADIUS server has a number of server configuration attributes which control almost all aspects of processing an incoming RADIUS request. Both the authentication and accounting RADIUS requests can use these server configuration attributes. By setting the appropriate values for these attributes, the system administrator can append attributes to the request, re-write any attribute of the request, proxy or replicate the request to another RADIUS server (based on criteria, not just the destination realm, i.e., ‘@realm’), choose an authentication method to use for a specific supplicant, etc. However, most server configuration is limited to *authentication* aspects only. [45] Note that this emphasis is important as our extension of the server concerns extending **authorization**, rather than authentication.

3.2.1.1.3 Selecting a particular configuration

The FreeRADIUS server can select its configuration based on any criteria listed in **Table 3-1**. [45]

Table 3-1: The criteria for the FreeRADIUS server to select a configuration

Attributes which have a given value
Attributes which do not have a given value
Attributes which are in the request (independent of their value)
Attributes which are not in the request
String attributes which match a regular expression
Source IP address of the request. This can be different than the NAS-IP-Address attribute.
Group of NAS boxes. (These may be grouped based on Source IP address, NAS-IP-Address, or any other configuration)
Integer attribute which match a range (e.g. <, >, <=, >=)
User-Name
A DEFAULT configuration
Multiple DEFAULT configurations

3.2.1.1.4 Authentication methods

The FreeRADIUS server supports a wide range of authentication types, such as: clear-text password in local configuration file (PAP), encrypted password in local configuration file, CHAP, MS-CHAP, MS-CHAPv2, authentication to a Windows Domain Controller (via ntlm_auth and winbindd), EAP with embedded authentication methods (like: EAP-MD5, Cisco LEAP, etc.), a Perl script, etc. [45]

3.2.1.1.5 Authorization methods

The FreeRADIUS server supports following authorization types as shown below: Local files, Local DB/DBM database, LDAP, a locally executed program. (similar to a CGI program), Perl script, Python program, MySQL DB, PostgreSQL DB, Coracle SQL DB, any IODBC SQL DB, IBM’s DB2. [45]

3.2.1.1.6 Accounting methods

The accounting methods, such as: local ‘datail’ files, local ‘wtmp’ and ‘utmp’ files, proxy to another RADIUS server, replicate to one or more RADIUS servers, and SQL

(Oracle, MySQL, Postgre SQL, Sybase, IODBC, etc), are supported by the FreeRADIUS server. [45]

3.2.1.1.7 Dialup Admin Web Administration Interface

The FreeRADIUS server has a web administration interface which was based upon PHP4. The dialup_admin interface supports [45]:

- Users in an LDAP database
- Users and Groups in SQL database (MySQL or PostgreSQL)
- Create, test, delete, change personal information, check accounting and change dialup settings for a user
- Accounting Report Generator
- Bad Users facility to keep a record of users creating problems
- Online finger facility
- Test RADIUS server
- Online Usage Statistics

3.2.1.1.8 Scripting Languages

The FreeRADIUS has plug-in modules supporting Perl and Python. Both of these languages allow scripts to modify RADIUS requests and responses in a very efficient and simple manner. [45]

3.2.1.2 Current version 2.0.5 improvements and bug fixes

The newest FreeRADIUS **version 2.0.5** was released in June 7th this year. This new version focuses on increasing the stability of the FreeRADIUS. Compared with the former releases, **version 2.0.5** has some improvements in feature and a number of bug fixes-listed in **Table 3-2**. [46]

Table 3-2: Version 2.0.5 improvements and bug fixes

Feature improvements	Permit SQL authorize_reply_query to be empty.
	Allow setting response packet type in Post-Proxy-Type Fail handler.
	Added install-chown target to set correct permission and ownership make RADMIN=radmin RGROUP=radius install-chown.
	Support for LDAP-Group and other dynamic comparison attribute in unlang.
	Added chroot support.
	Allow clients of 0/0.
	Moved many module configurations into raddb/modules/*.
Bug fixes	Allow proxying to virtual servers for accounting packets, too.
	Added num_fields function to PostgreSQL client. This lets clients be read from a PostgreSQL database.
	Updated proxy fallback mechanism to validate fallback servers, and to process fallback requests in a child thread.
	The realm module returns ok for LOCAL realms, not noop.
	Fixed some DHCP code handling. The examples should now work.

3.2.2 Proposed improvement to AAA

Traditionally an AAA protocol is used to make a binary **authentication** choice: allow a user access to a resource or not. This means that the authentication server sends to the supplicant a binary signal (effectively 1 or 0, on or off, allow or disallow). For example, in the case of an IEEE 802.11i WLAN access point which implements authentication, the access point acts as an authenticator and only allows the user to send frames for AAA until the authentication and authorization is successful. While the Diameter protocol has improved the RADIUS protocol in many aspects; Diameter still only supports a binary decision regarding authorization. In this thesis, we will explore the proposal for a non-binary authentication solution and implement it using in FreeRADIUS. A similar extension could be made to Diameter if desired, this remains as future work.

As FreeRADIUS can work in different network environments, for the purpose of having a concrete scenario for this thesis – we consider the use of this extended RADIUS together with IEEE 802.1x (a port based authentication mechanism has introduced in **section 2.2.2**). Like all use cases of AAA protocols, in the case of IEEE 802.1x, there are also three main entities: a supplicant (which is the client), an authenticator (which may be co-located with the access point), and the authentication server (which is the FreeRADIUS server).

The Extensible Authentication Protocol (EAP) is used in IEEE 802.1x to support several different authentication methods, such as: EAP-TLS (Transport layer Security), EAP-MD5 (Message Digest 5), and so on. EAP is used between the supplicant and the authentication server. This is an important change from the case of the NAS which we examined earlier, as only the NAS and the authentication server communicated. This additional entity and this additional protocol complicate the process as compared to the simpler NAS use case.

Figure 3.1 illustrates the IEEE 802.1x use case with binary authentication. Before the authentication succeeds, only an EAP message is acceptable for forwarding via the authenticator (access point); and until the authentication process is successful all the other types of packets can not be sent or if they are sent, then all other types of packets will be filtered out by the authenticator (see **Figure 3.1**). After the authenticator filters the messages received, each EAP message is placed into a RADIUS packet and sent to the FreeRADIUS server.

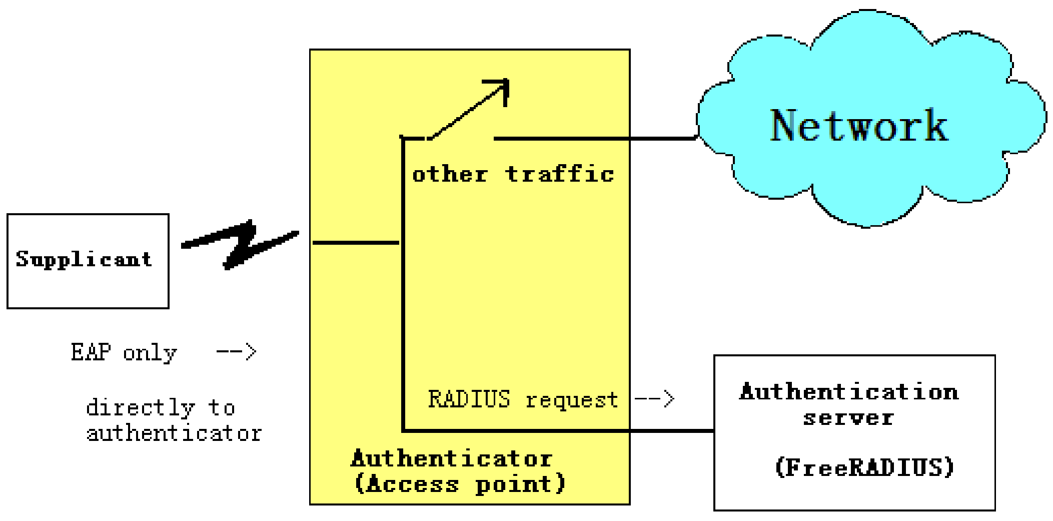


Figure 3.1 IEEE 802.1x use case of RADIUS and EAP

3.2.2.1 Non-binary solution of FreeRADIUS

In the traditional AAA process, the supplicant needs to communicate with an access point first to perform an IEEE 802.11 Associate operation. Once this has occurred the supplicant begins the authentication process using EAP with the authenticator. Only after the supplicant has received the EAP-success message, can it start sending frames outside of the local sub network (cell); i.e., all frames other than EAP frames will be dropped by the access point. The authentication process is shown in **Figure 3.2**:

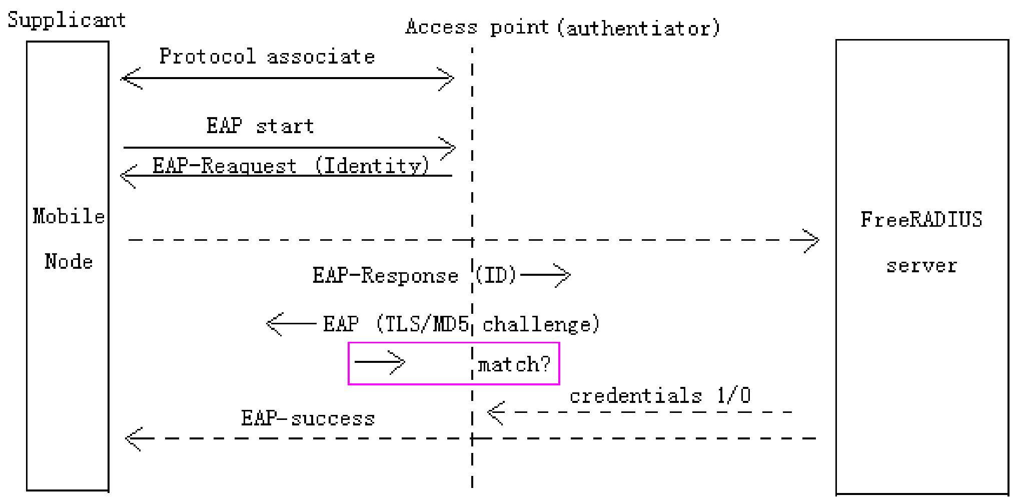


Figure 3.2 802.1x authentication process

From **Figure 3.2,** after the authentication is successful, the FreeRADIUS server will sends a message to the authenticator containing logically 1/0. By 1, it means the authentication is successful; while 0, means that the authorization request failed and the supplicant can not access the resource (in the case the network shown in **Figure 3.1**). In reality the message does not simply contain a one or zero, but is an `Access-Accept` or `Access-Reject` message (respectively); but as noted before this in a binary **authorization** result. This kind of binary authentication is the only authentication response which can be returned by RADIUS currently. Our proposed extension of the RADIUS protocol and our implementation in FreeRADIUS will provide a response to the Authenticator that allows it to control of the supplicant's bandwidth. This is illustrated with the image of a variable valve in **Figure 3.3 (b)**, rather than the switch shown in **Figure 3.2 (a)**.

In the proposed extension, rather than simply telling the access point to allow or deny a supplicant, the extended RADIUS response will tell the access point how much traffic a supplicant can send during a unit time. In contrast when the current binary response is sent to the Authenticator, all the supplicants generally have the same (maximum) bandwidth. However, the proposed non-binary solution gives the

FreeRADIUS server the ability to assign the supplicant a given amount of bandwidth, limit the supplicant to a certain maximum data rate, limit the supplicant to a number of packets, … . The server can even tell the supplicant how much traffic it is allowed. This later could be further extended to allow the supplicant to negotiate with the extended server to have more resources[2].

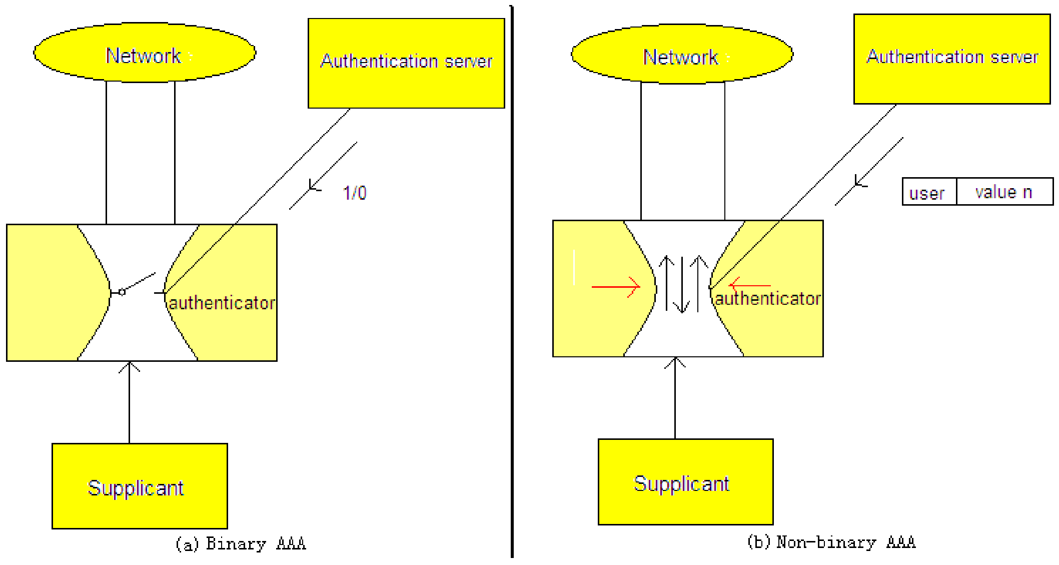


Figure 3.3 Comparison the binary and non-binary AAA solutions

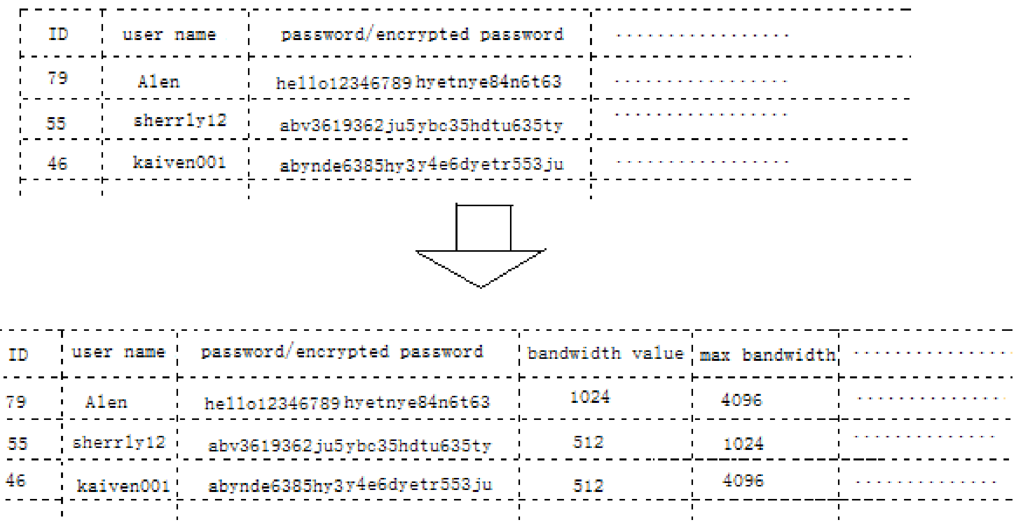
Figure 3.3 shows a comparison of the binary and non-binary solutions. As can be seen, the proposed solution replaces the switch inside the authenticator (which might be an access point) with more complex traffic control – based upon the response from the authentication server. This can be used by the authenticator to separately control the bandwidth (or other traffic parameters) for each supplicant.

On the authenticator side, a new traffic shaper will be added. This traffic shaper will implement traffic control as per the message from the authentication server. The details of this traffic shaper and the authenticator are described by Guo Jia in his thesis [4].

From the view of the authentication server, FreeRADIUS should be extended to send

[2] This idea of indicating to the supplicant how much of the resource is available to it seems to be applicable to not only the case of WLAN access but also to the case of fixed access. The later might be applicable for use in situations where the user's ISP sets a cap on usage, but currently has no **direct** means of communicating this limit or the user's remaining quantity of use - before the cap is applied. Comcast has proposed sending the user e-mail when they have exceed their cap of 250Gbytes/month; but if the user fails to observe this cap they risk having their service cut off for a year! Thus it would be useful to the user if they could learn how much remains to be used for this month - in order to avoid exceeding the cap; otherwise the user has to do their own accounting for their usage, but since the accounting records are only sent to the AAA server, this means they will have to duplicate this accounting.

a message containing the bandwidth. This requires that the FreeRADIUS client database be extended with a new column, as not only must the database row contain a user name or ID and the corresponding password information, but the row must also have a specification of the traffic parameters to be assigned to this user. Note that the table could be abbreviated; with each user's entry simply containing a default bandwidth and maximum bandwidth. The database could be extended like shown in **Figure 3.4**.



ID	user name	password/encrypted password	....
79	Alen	hello12346789 hyetnye84n6t63	....
55	sherrly12	abv3619362ju5ybc35hdtu635ty	....
46	kaiven001	abynde6385hy3y4e6dyetr553ju	....

ID	user name	password/encrypted password	bandwidth value	max bandwidth	....
79	Alen	hello12346789 hyetnye84n6t63	1024	4096	....
55	sherrly12	abv3619362ju5ybc35hdtu635ty	512	1024	....
46	kaiven001	abynde6385hy3y4e6dyetr553ju	512	4096	....

Figure 3.4 Database in authentication server

The bandwidth value in the database is the default value for the supplicant if the authentication is successful. This value will be sent by the FreeRADIUS server to the Authenticator (access point), and this value will be used to configure the traffic shaper to provide the specified bandwidth for this authenticated supplicant. Prior to the Authenticator receiving the bandwidth value from the extended RADIUS server, it might limit the user to a small bandwidth or only certain sizes of packets.

The maximum value in the database is used to define the maximal bandwidth for each supplicant. It should be the value the supplicant decided upon when it subscribed to the service and this subscription is registered in this FreeRADIUS server. In this thesis, we will provide a chance for the supplicant and the RADIUS server to negotiation how much bandwidth the supplicant would like to use for different situations. This maximum value limits the maximal bandwidth that the RADIUS server could specify for this supplicant.

The FreeRADIUS server sends the bandwidth value to both the access point **and** the supplicant when the authentication is successful. Subsequently the user might wish to negotiate for more bandwidth; in this case the process will occur as shown in **Figure 3.5**. Note that in **Figure 3.5** both the authenticator and supplicant are told the new bandwidth value.

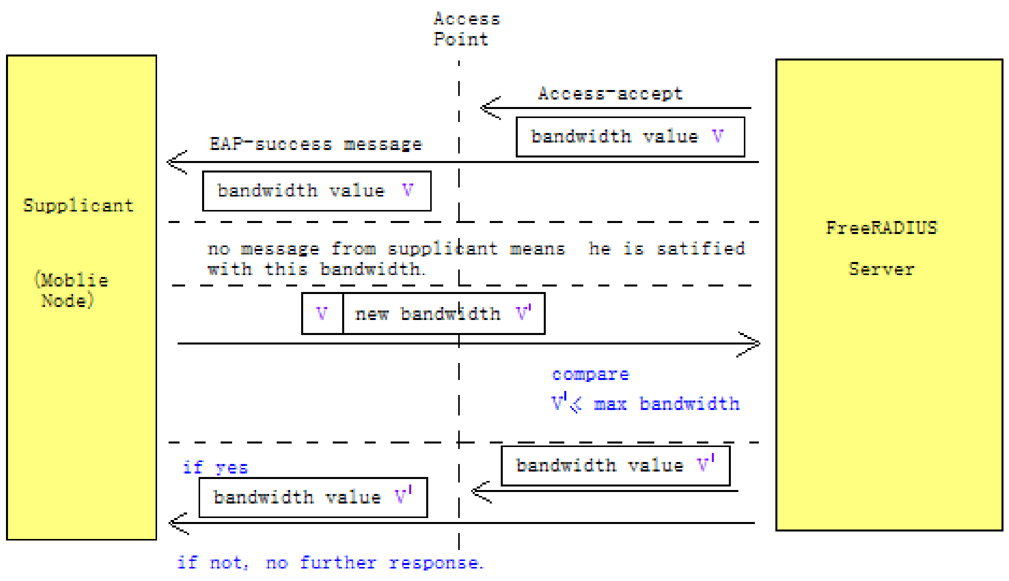


Figure 3.5 Bandwidth discussing process

3.2.2.2 New message structure for non-binary solution

Although the new solution need not change any part of the **authentication** process, the message indicating authentication success is not the same as in the binary authentication solution. That means more information should be contained in the success message, or we would say the new message structure should be established. Additionally, there needs to be a new message for bandwidth negotiation. These new messages should be acceptable and readable for both the authenticator (access point) and the supplicant. Otherwise, the supplicant and the authenticator will not able to leam the bandwidth information nor perform the bandwidth control (traffic shaping). Note that the same success message should be sent to the authenticator - even if the authenticator is not capable of performing traffic shaping; this enables backward compatibility and provides a smooth upgrade path (while minimizing administrative tasks when the authenticator is replaced with one which is extended to allow non-binary authentication). This means that when a supplicant receives this bandwidth information from the authentication server, it simply represents a potential maximum bandwidth which the authentication server has authorized - it does not mean that the authenticator can actually provide this bandwidth! (Note that this suggests that the resulting system supports non-binary authentication and variable resource authorization.)

3.2.2.2.1 Authentication success message

All the messages sent out from the authentication server are in the same **RADIUS format** as shown in **Section 2.2.2**. The authenticator (access point) will reformat the information into suitable formats for different situations.

The messages sent to the authenticator (access point) are the `Access-Reject` or `Access-Accept` message. For a traditional RADIUS packet, no bandwidth information is contained in these messages. For the non-binary solution, bandwidth information should be added in the `Access-Accept` message, with a new attribute field in the RADIUS packet structure which contains the bandwidth value. The `EAP-Success` message is also encapsulated in the `Access-Accept` message and will forward by the authenticator to the supplicant. This `EAP-success` message also should be extended to convey the bandwidth information to the supplicant.

The supplicant will receive an EAP success/failure message. All the EAP messages are in the same format (shown in **Figure 3.6**).

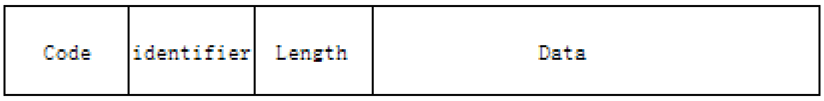


Figure 3.6 EAP message format

The code field will indicate the message type. Code value 3 or 4 means success or failure in this field. In the binary authentication scheme the Success and Failure messages are short and do not require any data in binary authentication field. These messages simply indicate the result of the authentication server's decision.

In the new non-binary authentication, the failure message does not need to be changed. But the success message should contain the supplicant's bandwidth information. That means the `EAP-success` message should contain the bandwidth value in data portion of the message expressed like an attribute. Thus when the `EAP-success` message arrives at the supplicant, the supplicant learns its allowed bandwidth.

3.2.2.2.2 Bandwidth negotiation

Bandwidth negotiation is a totally new part of the non-binary authentication process. After authentication successful, if the supplicant wishes to change the maximum bandwidth he or she is able to use, then negotiation can be done with the RADIUS server. From the view point of the authenticator (access point), negotiation is not limited to EAP messages, since the supplicant has been successfully authenticated any type of packet could be used to carry the negotiation message. Thus either the supplicant (now simply a client) directly sends a RADIUS request or the authenticator must receive the message and reformat it into a negotiation message which is send as a RADIUS message to the FreeRADIUS server. In this thesis we propose a new attribute value should be defined for use by the supplicant, authenticator, and the FreeRADIUS server for use in a negotiation message.

The FreeRADIUS server would be extended to provide this new bandwidth negotiation function. When the negotiation message arrives at the server, the server will know that this is a negotiation message because a new code will be assigned for this message (it is proposed that the code can be 250-253 (these codes reserved for experimental use) – a request has not been made to IANA for this code, but his should be done as part of future work). The negotiation message not only contains the request bandwidth, but also the suppliant information. So the function will used the supplicant information, like: the same identifier that was used to identify the relevant client in the client database, to determine the maximum bandwidth in the database. Then a comparison will make between the maximum bandwidth and the requested bandwidth: if the maximum value is smaller than the requested one, the bandwidth request will be ignored; otherwise, the requested bandwidth will be sent in a message to both the authenticator (access point) and the supplicant to notify them that the maximum bandwidth should be changed to this requested value. This is should be an extended RADIUS message contains the new bandwidth with a special code (for example: code 252 can be used to notice this is a bandwidth update message).

3.3 Implementation of these improvements

The implementation of the improved FreeRADIUS can be divided into three parts:

- ✧ Change the existing SQL database, to add the default bandwidth value and maximum bandwidth value for each supplicant (client);
- ✧ Extend the authentication success message, to include the bandwidth value as an attribute;
- ✧ Add the negotiation function to the FreeRADIUS program; make this negotiation between the supplicant and the FreeRADIUS server work.

3.3.1 Modifying the database

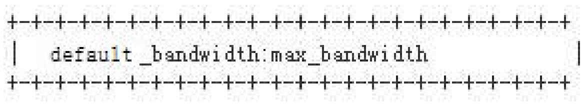
After FreeRADIUS is installed in the server, the first thing to do is perform the configuration, then setup the SQL database. This is because we decided to authenticate users based upon an entry in an SQL database. MySQL is select in this FreeRADIUS as it is one of the most widely used SQL database. This method of configuration was chosen because it allowed us to create, manage and control the user database in the FreeRADIUS server easily. The traditional MySQL database contains a user name or ID for each client. Different attribute values have different meanings, for example, “2” means that the attribute contains a user-password. For the non-binary authentication, new columns should be added to the client database to define the default bandwidth value and maximum bandwidth for each client. Several different approaches could be used to implement these extensions. One approach is to re-use an existing attribute for these two new values; while the other approach is to add new attributes for the RADIUS database. Both the solutions have their own advantages and disadvantages, as shown in **Table 3-3:**

Table 3-3: Comparison of the two alternatives for modifying the FreeRADIUS database

	1. Use the old attributes	**2. Add new attributes**
Advantage	Using existing attributes will reduce the work.	Two new attributes are created especially for these two bandwidth values; they will not influence any other parts of the database.
Disadvantage	An attribute has different meanings depending upon the context of use. If not choose well, this could impact other parts of FreeRADIUS.	Adding the new attributes in RADIUS protocol could affect a lot of parts of the FreeRADIUS server and require a lot of work to make it running well.

In this thesis, the first solution was used initially to perform an experiment to see how non-binary authentication might work. After this experiment we used the second approach to implement the non-binary solution.

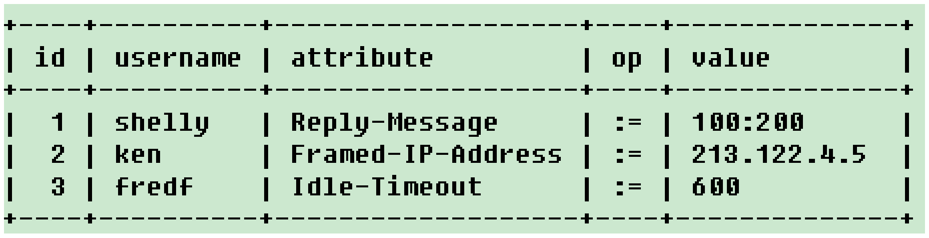
In MySQL, the command: “insert” can be used to add new information in the database. As we started with the first approach, we choose to modify the “Reply-message” for the experiment. A list separator (“:”) is used in the attribute value part to separate different items. So adding a bandwidth value and the maximum bandwidth was straight forward. The stored format is shown in **Figure 3.7**:



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   default _bandwidth:max_bandwidth          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.7 The new value stored format

Using the command "insert into radreply (id, username, attribute, op, value) values(1, fredf, Reply-Message, :=, 100:200); ", we can check the database and see this result in **Figure 3.8**.



```
+----+----------+--------------------+----+---------------+
| id | username | attribute          | op | value         |
+----+----------+--------------------+----+---------------+
|  1 | shelly   | Reply-Message      | := | 100:200       |
|  2 | ken      | Framed-IP-Address  | := | 213.122.4.5   |
|  3 | fredf    | Idle-Timeout       | := | 600           |
+----+----------+--------------------+----+---------------+
```

Figure 3.8 FreeRADIUS MySQL database

This experiment shows it is easy to add new information to the MySQL database. The value can retrieve by using the attribute. To achieve the non-binary authentication, new attributes should be defined. There are two ways we can use to define the new attributes: one is to define two new global attributes, the other is to define vendor specific attributes. In the following subsections we will introduce the two approaches separately.

3.3.1.1 Define global attributes

These attributes are defined in the file "radius.h" of the "src" files. As the attribute values 192-223 are reserved for experimental use to add new attributes, we will use the value 193 for the default bandwidth and 194 for the maximum bandwidth. The modified code is as shown below:

```
// modified part of radius.h

/* here we define our own attributes, which are Default-Bandwidth and Max-Bandwidth.
We allocate them from the reserved for experimental use attribute numbers.
*/
#define PW_DEFAULT_BANDWIDTH            193
#define PW_MAX_BANDWIDTH                194
```

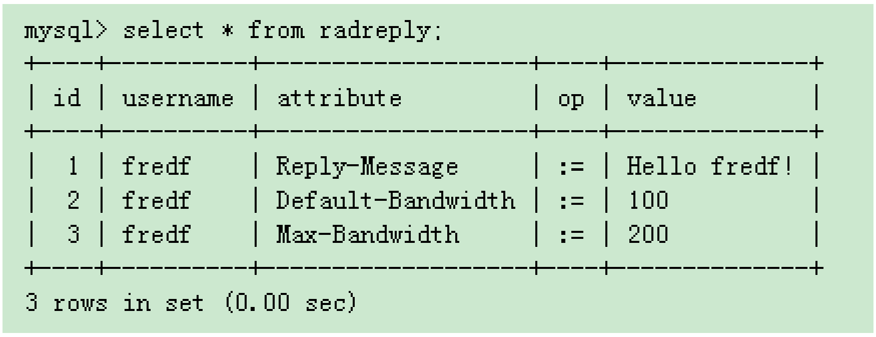
We also need to modify the dictionary file as shown below:

```
// modified part of /etc/raddb/dictionary

# New Attributes are defined here.
# The values 192-223 were reserved for experimental use, we will use two of these values.
ATTRIBUTE       Default-BandWidth       193     string
ATTRIBUTE       Max-Bandwidth           194     string
```

The dictionary file should be contained in the library.

```
// modified part of /usr/local/etc/raddb/dictionary

$INCLUDE        /etc/raddb/dictionary
```

After the definition of the two attributes, the new MySQL database entry for the user “fredf” is shown in **Figure 3.9**.



```
mysql> select * from radreply;
+----+----------+-------------------+----+--------------+
| id | username | attribute         | op | value        |
+----+----------+-------------------+----+--------------+
|  1 | fredf    | Reply-Message     | := | Hello fredf! |
|  2 | fredf    | Default-Bandwidth | := | 100          |
|  3 | fredf    | Max-Bandwidth     | := | 200          |
+----+----------+-------------------+----+--------------+
3 rows in set (0.00 sec)
```

Figure 3.9 The new SQL database

3.3.1.2 Define vendor specific attributes

The first thing to do is creating a special dictionary for vendor specific attributes, the dictionary is created in: /usr/local/share/freeradius/dictionary.kth.

```
# -*- text -*-
##############################################################################
#
#    KTH dictionary
#
# 2008.09.01 GQMJr for experimental use
#
##############################################################################

VENDOR        KTH            933

BEGIN-VENDOR    KTH

ATTRIBUTE    KTH-default-bandwidth        3     integer
ATTRIBUTE    KTH-max-bandwidth            4     integer

END-VENDOR    KTH
```

The newly established dictionary should be included in the FreeRADIUS dictionary:

```
# include the dictionary to /usr/local/share/freeradius/dictionary
    $INCLUDE dictionary.kth
```

Then the new attributes have been defined and the MySQL database can use them as attributes to store users' information.

Using these new attributes, we can specify for each user a default bandwidth and maximum bandwidth value. Now that we have extended the set of attributes which are available, it is now time to examine the new authentication success message.

3.3.2 New authentication success message

An authentication success message is sent to both the access point (in the format of a RADIUS `Access-Accept` message) and the supplicant (in the form of a RADIUS encapsulated EAP success message) when authentication is successful. As the database has already been extended; the main problem was how to add the bandwidth value to the authentication success message(s).

For the `EAP-Success` message in a binary authentication server, only a code, identifier, and length exist. As an EAP message has space for data following the length field, in non-binary authentication the `EAP-Success` message will simply be extended to contain the bandwidth information.

From the view of the authentication server, all the messages sent are RADIUS format messages. For the purpose of telling the authenticator (access point) how much bandwidth the supplicant can use, an attribute field containing the bandwidth was added.

To do this, for testing purposes the "radiclient.c" source code should be modify to include the default bandwidth and the maximum bandwidth in the `access-accept` message.

```
// modified part of radclient.c

        /* handle DEFAULT bandwidth parameters */
        if (reply->code !=PW_AUTHENTICATION_REJECT) {
                VALUE_PAIR *vp;

                /* Find the pair with attribute PW_DEFAULT_BANDWIDTH*/
                vp = reply->vps;

                while(vp && vp->attribute != PW_DEFAULT_BANDWIDTH)
                        vp = vp->next;
                if( vp == NULL ) {
                        printf("The packet received does not contain DEFAULT bandwith
information\n");
                        goto packet_done;
                }
                else {
                /* Extract the bandwidth information from received packet.
                 */
                        int  default_bandwidth;
                        char buf[128];
                        /* Here 128 was used to make the buffer big enough, but not to
much redundant.*/

                        strcpy(buf, vp->vp_strvalue);
                        default_bandwidth = atoi(buf);

                        printf("The DEFAULT bandwidth information recevied is:\n");
                        printf("default_bandwidth: %d\n", default_bandwidth);

                }
        }

        /* handle MAX bandwidth parameters */
        if (reply->code !=PW_AUTHENTICATION_REJECT) {
                VALUE_PAIR *vp;

                /* Find the pair with attribute PW_MAX_BANDWIDTH*/
                vp = reply->vps;

                while(vp && vp->attribute != PW_MAX_BANDWIDTH)
                        vp = vp->next;

                if( vp == NULL ) {
                        printf("The packet received does not contain MAX bandwith
information\n");
                        goto packet_done;
                }
                else {
                /* Extract the bandwidth information from received packet.
                 */
                        int  max_bandwidth;
                        char buf[128];

                        strcpy(buf, vp->vp_strvalue);
                        max_bandwidth = atoi(buf);
                        printf("The MAX bandwidth information received is:\n");
                        printf("max_bandwidth: %d\n", max_bandwidth);

                }
        }
```

We will also modify the “eap.c” in the file of /src/modules/rlm_eap. This will add the bandwidth information into the EAP success message

```
  /*
        *       EAP-Message is always associated with Message-Authenticator but not
vice-versa.
        *
        *       Don't add a Message-Authenticator if it's already there.
        */
        vp = pairfind(request->reply->vps, PW_MESSAGE_AUTHENTICATOR);
        if (!vp) {
                vp = paircreate(PW_MESSAGE_AUTHENTICATOR, PW_TYPE_OCTETS);
                memset(vp->vp_octets, 0, AUTH_VECTOR_LEN);
                vp->length = AUTH_VECTOR_LEN;

                pairadd(&(request->reply->vps), vp);
        }

        /* only output the reply attribute when reply code is PW_EAP_SUCCESS */
        if ((!request->reply->code) && (reply->code != PW_EAP_SUCCESS)) {
                pairdelete(&request->reply->vps, PW_DEFAULT_BANDWIDTH);
                pairdelete(&request->reply->vps, PW_MAX_BANDWIDTH);
                pairdelete(&request->reply->vps, PW_KTH_Default_Bandwidth);
                pairdelete(&request->reply->vps, PW_KTH_Max_Bandwidth);
                vp_printlist(stdout, request->reply->vps);
        }
```

This configure make the `EAP-Success` message contains the default bandwidth value and maximum bandwidth. If the authentication, the bandwidth information will not be contained in the fail message.

3.3.3 Bandwidth negotiation function

The new bandwidth negotiation function enables the FreeRADIUS server to respond to RADIUS client requests for a different maximum bandwidth. After the authentication has succeeded, the supplicant can send message to the FreeRADIUS server to ask for a different maximum bandwidth.

The elements of this negotiation function are complex. Therefore, actually implementing this in the FreeRADIUS code was more difficult. All the messages sent to and from authentication server are in the RADIUS message format, with a code field that labels the message type. In order to process the negotiation properly, a new message type should be defined. As we discussed in **section 3.2.2.2.2** in this thesis, the code field 250-253 are reserved for experimental use, we can define 251 as the `bandwidth-negotiation-request` message and 252 as the `bandwidth-negotiation-accept` message. Then when the message with code number 251 arrives at the FreeRADIUS server, the main function will detect it is a bandwidth negotiation message and send the data in the message to the negotiation function to process.

When a message was recognized as the bandwidth negotiation message, information will go to the negotiation function to continue the working process. Functions will work as shown below:

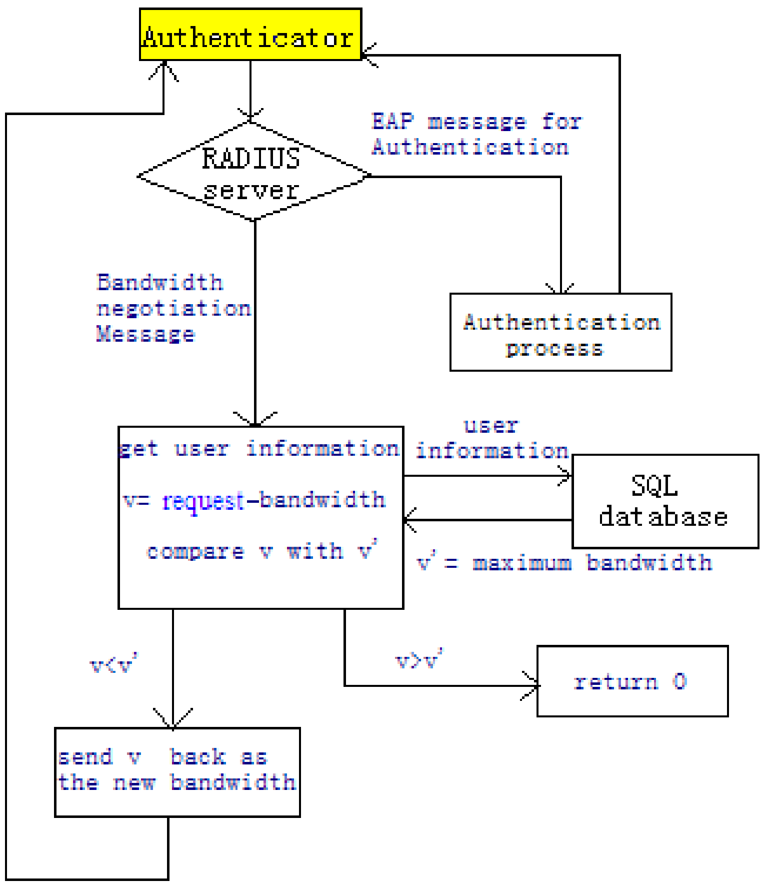


Figure 3.10 Negotiation process

The codes for the bandwidth negotiation process are here, but not yet combined with the main FreeRADIUS function.

```
//the bandwidth negotiation function

int band_width (int user_name int request_width)
{
    int max_width;
    max_width = value_sql (user_name, attribute value=194);

    if (request_width > max_width)
    {
        return 0;
    }
    else
    {
        return request_width;
    }

}
```

The bandwidth negotiation process is very clear and not difficult, but how to make both the supplicant and the FreeRADIUS server handle these messages is more complex than I thought. However, an attribute for the negotiation message has been defined and the database extended to store the required value (per client). This new attribute must be acceptable to the supplicant, FreeRADIUS server, and the authenticator (access point). As the code to actually implement these messages has not yet been added to the RADIUS server, I have not be able to test that this new attribute will be ignored by existing supplicants and authenticators; nor that it provides the correct information to the extended supplicant and authenticator.

Chapter 4: Analysis of the proposed extensions to FreeRADIUS

In this thesis, we extended the FreeRADIUS server, changing authentication from binary to non-binary and introduced bandwidth negotiation. As we have only implemented some parts of this extension in the FreeRADIUS server, experiments were made to test the new authentication method. A comparison was also made between this new method and an unmodified FreeRADIUS server.

4.1 Experimental results

As a part of the non-binary authentication research, a real test for the new system would require all the parts of the network to operate together. However, as Guo Jia’s authenticator part and Zhang Hengchong’s supplicant part are not yet finished, it is impossible to test all the three non-binary authentication entities working together. In this thesis, all the experiments were performed only with the FreeRADIUS server part and test application (which can act as a RADIUS client).

The extension of the FreeRADIUS server can be divided into two parts: the non-binary authentication part and the bandwidth negotiation part. For the authentication part, there is a “radtest” application provided with the FreeRADIUS source code that was adapted (as described in **section 3.2.2**) for my experiments. As the bandwidth negotiation part has only been design and not implemented, we can not do any experiments concerning it.

We have conducted two experiments to test the non-binary authentication server. The first test is based on the non-binary authentication method without a new attribute; while the other is based on the non-binary authentication method with new attributes. For the second test, experiments are made to test both authentication success and authentication fail.

4.1.1 Experiment on non-binary RADIUS server without new attributes

In this experiment, we used the existing attribute; with application “radtest”, we could get the default bandwidth and the maximum bandwidth information in a “reply-message”. As a reply- message is always sent as a response to the supplicant, our use of the “reply-message” attribute will have minimal affect on the FreeRADIUS server. The experiment result is shown in **Table 4-1**.

Table 4-1 Authentication success results (without any new attributes)

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main # ./radtest
        shelly hello localhost 1812 testing123
Sending Access-Request of id 140 to 127.0.0.1 port 1812
         User-Name = "shelly"
         User-Password = "hello"
         NAS-IP-Address = 127.0.0.2
         NAS-Port = 1812

rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=140,
        length=29
         Reply-Message = "100:200"

The bandwidth information recevied is:
default_bandwidth: 100
max_bandwidth: 200
```

The default bandwidth and maximum bandwidth can be sent when the authentication succeeds. No further experiments are made regarding this, as we just used this result to see if the non-binary authentication is realizable.

4.1.2 Experiment on non-binary RADIUS server with new attributes

In this part, experiments are made on the server after it had been extended with the two new attributes (for the default bandwidth value and maximum bandwidth value). These experiments test whether these parts are functioning well. We use a valid user name and password at first to perform a test whose results should be the authentication success message, and then we use a valid user name but an incorrect password to check the authentication fail case.

For testing of a supplicant or authenticator we have made a simple application that can send the required messages with the additional attributes. What remains is to actually implement the code which supports adding these attributes to the proper RADIUS messages and to implement the bandwidth negotiation function.

The results of using the “radtest” command when using a correct user name and password is shown in **Table 4-2**. It is the RADIUS `Access-accept` message.

Table 4-2 Authentication success results (with new attributes) by "radtest" command

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main      ./radtest
          fredf fredf localhost 1812 testing123
Sending Access-Request of id 214 to 127.0.0.1 port 1812
          User-Name = "fredf"
          User-Password = "fredf"
          NAS-IP-Address = 127.0.0.2
          NAS-Port = 1812

rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=214,
          length=68
          Reply-Message = "Hello fredf!"
          Default-BandWidth = "100"
          Max-Bandwidth = "200"
          KTH-default-bandwidth = 3
          KTH-max-bandwidth = 4

Received response ID 214, code 2, length = 68
          Reply-Message = "Hello fredf!"
          Default-BandWidth = "100"
          Max-Bandwidth = "200"
          KTH-default-bandwidth = 3
          KTH-max-bandwidth = 4
          name: Reply-Message        attribute: 18     vendor: 0
          name: Default-BandWidth attribute: 193    vendor: 0
          name: Max-Bandwidth        attribute: 194    vendor: 0
          vendor: KTH    name: KTH-default-bandwidth    attribute: 3    vendor: 933
          vendor: KTH    name: KTH-max-bandwidth      attribute: 4       vendor: 933

    The DEFAULT bandwidth information recevied is:
    default_bandwidth: 100
    The MAX bandwidth information received is:
    Max_bandwidth: 200
```

We also use the "eaptest" to see the result of the EAP message, and the results are showed in **Table 4-3**. It is the `EAP-success` message.

Table 4-3 Authentication success results (with new attributes) by "eaptest" command

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main # ./eaptest
          fred fred
+++> About to send encoded packet:
    User-Name = "fredf"
    Cleartext-Password = "fredf"
    EAP-Code = Response
    EAP-Id = 210
    EAP-Type-Identity = "fredf"
    Message-Authenticator = 0x00

<+++ EAP decoded packet:
    Reply-Message = "Hello fredf!"
    EAP-Message = 0x01d300160410215716a8606857fd387004c142548c70
    Message-Authenticator = 0x7efaaa66b147e2b35ad0a6c21ff61135
    State = 0x05de8aa1050d8ebbd18207d5bd212ccc
    EAP-Id = 211
    EAP-Code = Request
    EAP-Type-MD5 = 0x10215716a8606857fd387004c142548c70

+++> About to send encoded packet:
    User-Name = "fredf"
    Cleartext-Password = "fredf"
    EAP-Code = Response
    EAP-Id = 211
    Message-Authenticator = 0x00000000000000000000000000000000
    EAP-Type-MD5 = 0x102cd41af3a1c18cc935aea4ef7274f1d6
    State = 0x05de8aa1050d8ebbd18207d5bd212ccc

<+++ EAP decoded packet:
    Reply-Message = "Hello fredf!"
    Default-BandWidth = "100"
    Max-Bandwidth = "200"
    KTH-default-bandwidth = 3
    KTH-max-bandwidth = 4
    EAP-Message = 0x03d30004
    Message-Authenticator = 0xc3a8dfc9d623f39f293137a5d98e6fdc
    User-Name = "fredf"
    EAP-Id = 211
    EAP-Code = Success
```

This experiment shows the result when the authentication is successful, thus the default bandwidth and maximum bandwidth will be contained in the reply message.

This result shows that our solution works correct.

We also made a test using a valid user name but an incorrect password to see that authentication fail message is sent. This is the `Access-reject` message

Table 4-4 Authentication results when password is incorrect by "radtest" command

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main # ./radtest
        fredf nikos localhost 1812 testing123
Sending Access-Request of id 26 to 127.0.0.1 port 1812
    User-Name = "fredf"
    User-Password = "nikos"
    NAS-IP-Address = 127.0.0.2
    NAS-Port = 1812

rad_recv: Access-Reject packet from host 127.0.0.1 port 1812, id=26,
        length=34
     Reply-Message = "Hello fredf!"
     Received response ID 26, code 3, length = 34
     Reply-Message = "Hello fredf!"
     name: Reply-Message      attribute: 18    vendor: 0

The packet received does not contain DEFAULT bandwith information
The packet received does not contain MAX bandwith information
```

We can see clearly that when authentication fails, no bandwidth information will be sent. That is just what we believed that the message would like.

An experiment on "eaptest" was also made to see the fail message of the authentication. As shown in **Table 4-5**, no bandwidth information will come out in the fail message.

Table 4-5 Authentication results when password is incorrect by "eaptest" command

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main # ./eaptest
          jia 12345
+++> About to send encoded packet:
    User-Name = "jia"
     Cleartext-Password = "12345"
     EAP-Code = Response
     EAP-Id = 210
     EAP-Type-Identity = "jia"
     Message-Authenticator = 0x00

<+++ EAP decoded packet:
     EAP-Message = 0x01d30016041050d5a5903e3909bbbd8b243f0e32c046
     Message-Authenticator = 0x7d2be2347c0665e73cfdf98964c41584
     State = 0xc17a5dbdc1a95900978f167a5e24092b
     EAP-Id = 211
     EAP-Code = Request
     EAP-Type-MD5 = 0x1050d5a5903e3909bbbd8b243f0e32c046

+++> About to send encoded packet:
     User-Name = "jia"
     Cleartext-Password = "12345"
     EAP-Code = Response
     EAP-Id = 211
     Message-Authenticator = 0x00000000000000000000000000000000
     EAP-Type-MD5 = 0x1085a7272f27adac8d71c8d1daf6995136
     State = 0xc17a5dbdc1a95900978f167a5e24092b

<+++ EAP decoded packet:
     EAP-Message = 0x04d30004
     Message-Authenticator = 0x0ebf5331dc6aed18ef9cfa86610fdaa9
     EAP-Id = 211
     EAP-Code = Failure
```

From these experimental results, the goal of non-binary authentication has been achieved.

This thesis project, all the passwords are stored as clear text, but the password encrypting arithmetic are still containing in the main code. For example, if the passwords want to encrypt by CHAP, there is a specially function working with them.

4.2 Comparing with other methods and improvements

In **section 2.5**, we compared FreeRADIUS and Diameter. The new AAA protocol Diameter has improved a lot of different functions in RADIUS and solved some problems. But as RADIUS is very widely used AAA server, it is unlikely that RADIUS will be replaced in a short period of time.

From the result of our experiment, we can see the authenticator (access point) can receive a default bandwidth of the supplicant together with the authentication success message. Our extended FreeRADIUS changed the binary authentication into a non-binary authentication. This is not simply change from 1/0 to a variable value, but a revolutionary change in authentication. While Diameter has solved some bugs and disadvantages of the RADIUS, it is obvious that non-binary authentication should also be added to Diameter.

The non-binary authentication method gives the authentication server the ability to control the bandwidth for each supplicant, and allows allocation of the total net bandwidth to be made in a more flexible manner. The newly proposed bandwidth negotiation process enables the supplicant request the bandwidth necessary for its current use. The result is that users of a FreeRADIUS server could indicate the amount of bandwidth which they actually need, rather than always being granted their maximum bandwidth; this might enable better sharing of resources.

Chapter 5: Conclusions

Although the server is not running together with the authenticator (access point) and the supplicant, the proposed extensions suggest that non-binary authentication might be possible and could offer some new possibilities to supplicants. This work also suggest that AAA protocols are moving into a new period where their use is not simply to authenticate users and to authorize their use of resource, but rather to allow more fine graded control of the use of this resource. Thus different supplicants can have their own bandwidth values and subsequently negotiate their maximum bandwidth with the FreeRADIUS server. That potentially makes the whole network environment working more effective. The accounting part of AAA protocol should also be extended in order to charge the supplicant more for requests to increase the maximum bandwidth.

In order for this non-binary authentication to become used all around the world will require a huge amount of work, and what we have done is first step. One of the core problems has been solved: new attributes have been defined specially for the default bandwidth and the maximum bandwidth, and these values could be sent out together with the authentication success message. While the results of this thesis project itself are poor, the ideas proposed represent a good starting point for further research, and we are sure that someone else will realize these ideas and enable an interesting new epoch for AAA systems.

Chapter 6:Future work

While RADIUS has developed a lot during the past few years, it still needs to be improved. How to modify the AAA protocol to operate efficiently while maintaining its security is a question all the researchers want to realize. That is also one of the most important future works for AAA improving.

In the world today, a group number of people are working on the improvement of the different AAA protocols. Such like Diameter has solved some disadvantage of RADIUS, a clear question remains: Should there be continued development of RADIUS or should this effort be put into its successor: Diameter. In this thesis we have chosen to only look at extending FreeRADIUS, thus applying these same extensions to Diameter should be part of any future work (as mentioned in **Section 3.2.2**).

For FreeRADIUS, we still have some suggestions for future work for making it into a non-binary authentication server:

The first step is to combine the bandwidth negotiation function with the main FreeRADIUS code, in order to realize this new function. Negotiation offers a great benefit for the supplicant and the internet service provider. With the negotiation process, if the supplicant has greater control of their data flows, then they do not need to pay for extra bandwidth which they did not use. The internet service provider can use this bandwidth to support other supplicants.

A judgment mechanism can also be added in the RADIUS server. For each supplicant in the database, there is a record value. The authenticator will send a signal to the RADIUS server in a unit of time to notice the server whether the supplicant is running “friendly”. If the supplicant always follows the maximum bandwidth it allows using, his record value would be rise; when the record value becomes big enough, the RADIUS server can encourage the supplicant by raising his maximum bandwidth in the system or raising his user level. If someone always tries to use more bandwidth than he is allowed, the record value will reduce; when the record value becomes low enough, the supplicant will receive some punishments: like reduce his maximum bandwidth in the database, or even take him as a viciousness user and forbid his account for a while. With this function, the RADIUS server will work in a more intelligent way.

The accounting component can also be improved to make use of the information about used & requested bandwidths. As we have achieved the non-binary authentication, the accounting should be modified to account for this variable bandwidth. Therefore not only time of usage needs to be recorded for each supplicant, but the packet flux should also be thought of and used as input to the charging computation.

Our improvement was designed to create a non-binary authentication mode for the FreeRADIUS server. As the Diameter becomes more widely used a non-binary Diameter might also be important to realize. There will also be a need to develop more cooperative and capable supplicants. We have given some suggestions for future work, but these suggestions are for the near future as they should be realized a short time. However, with this develop new problems are likely to appear, which have not been apparently given the binary nature of authentication thus far.

References

[1]. Wikipedia, "AAA protocol", http://en.wikipedia.org/wiki/AAA_protocol, Apr. 20th, 2008

[2]. Cisco, "Cisco Craft Works Interface Configuration Applications Reference Guide; Chapter 2: AAA Application", http://www.cisco.com/univercd/cc/td/doc/product/software/iosxr32/cwi_32/cwiap32/cw32caaa.pdf, Apr. 20th,2008

[3]. Yuanchao Zhao, Jian Chen, and Daoben Li, "AAA protocol- Diameter analyze", "China data traffic",http://news.ccidnet.com/art/2013/20041116/177981_1.html, Nov. 16th, 2004 16

[4]. Guo Jia, "non-binary authentication", Masters thesis, Department of Communication Systems, School of Information and Communication Technology, Royal Institute of Technology, Stockholm, Sweden, 2008 (draft)

[5]. Zhang Hengchong, "non-binary authentication—supplicant", Masters thesis, Department of Communication Systems, School of Information and Communication Technology, Royal Institute of Technology, Stockholm, Sweden, 2008 (draft)

[6]. James D. Carlson, "PPP Design, Implementation, and Debugging", Second edition, Addison-Wesley, 2000, ISBN 0-201-7 0053-0

[7]. Wikipedia, "RADIUS", http://en.wikipedia.org/wiki/RADIUS, May. 6th, 2008

[8]. Wikipedia, "RADIUS", http://zh.wikipedia.org/wiki/RADIUS, Jan 20th, 2008

[9]. Network Working Group, RFC 2085, "HMAC-MD5 IP Authentication with Replay Prevention", http://www.rfc-editor.org/rfc/rfc2085.txt, Feb. 1997

[10]. H3C (IToIP Solutions Expert), "AAA&RADIUS&HWTACACS Introduction", http://www.h3c.com/portal/Products___Solutions/Technology/Security_and_VPN/AAA_RADIUS_HWTACACS/200701/195605_57_0.htm, May 9th.2008

[11]. Wikipedia, "Diameter (protocol)", http://en.wikipedia.org/wiki/Diameter_%28protocol%29, May 8th, 2008

[12]. Wikipedia, "IPsec", http://en.wikipedia.org/wiki/IPSEC, May 8th, 2008

[13]. Wikipedia, "TLS", http://en.wikipedia.org/wiki/Transport_Layer_Security, May 8th, 2008

[14]. Wikipedia, "TACACS", http://en.wikipedia.org/wiki/TACACS, May 8th, 2008

[15]. Wikipedia, "TACACS+", http://en.wikipedia.org/wiki/TACACS%2B, May 8th, 2008

[16]. Vincent Chen, "Security control protocol- TACACS+",

http://www.cnitblog.com/wildon/archive/2007/03/19/24261.html, Feb. 7th, 2003

[17]. Network Working Group, RFC 2039, “Applicability of Standards Track MIBs to Management of World Wide Web Servers”, http://www.rfc-editor.org/rfc/rfc2039.txt, Nov. 1996

[18]. Network Working Group, RFC 2138, “Remote Authentication Dial In User Service (RADIUS)”, http://www.ietf.org/rfc/rfc2138.txt, Apr. 1997

[19]. Network Working Group, RFC 2865, “Remote Authentication Dial In User Service (RADIUS)", http://www.ietf.org/rfc/rfc2865.txt, Jun. 2000

[20]. Network Working Group, RFC 2866, “RADIUS Accounting”, http://www.ietf.org/rfc/rfc2866.txt, Jun. 2000

[21]. Microsoft TechNet, “RADIUS Packet Format”, http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/intwork/inbc_ias_wfwi.mspx?mfr=true, May 10th, 2008

[22]. HP website, “RADIUS Overview”, http://docs.hp.com/en/T1428-90025/ch01s01.html, May 10th, 2008

[23]. IEEE, “IEEE Standards for Local and Metropolitan Area Networks: Standard for Port Based Network Access Control”, IEEE Std 802.1x-2004, http://standards.ieee.org/getieee802/download/802.1X-2004.pdf, Oct. 2004

[24]. Wikipedia, “IEEE 802.1x”, http://en.wikipedia.org/wiki/802.1x, Aug. 11th, 2008

[25]. Network Working Group, RFC 3850, “IEEE 802.1x Remote Authentication Dial In User Service (RADIUS) Usage Guidelines”, http://www.ietf.org/rfc/rfc3580.txt, Sep. 2003

[26]. Tuomas Aura, Network security at UCL, Microsoft Research, UK, “Lecture 9: WLAN Security”, http://research.microsoft.com/users/tuomaura/Teaching/Lecture%2009%20-%20WLAN.pdf, Jan.-Feb. 2008

[27]. Wikipedia, “Open source”, http://en.wikipedia.org/wiki/Open_source, May 10th, 2008

[28]. Sara Dannerud, “Sårbarheter i routrar och switchar”, Batchelors Thesis, Department of Communications, School of Information and Communication Technology, Royal Institute of Technology (KTH), COS/CCS 2008-03, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080313-Sara_Dannerud-with-cover.pdf, Mar. 13th, 2008.

[29]. Wikipedia, “FreeRADIUS”, http://en.wikipedia.org/wiki/FreeRADIUS, May 10th,2008

[30]. FreeRADIUS, http://www.freeradius.org/, Apr. 30th, 2008

[31]. Bugzilla Bug, “GNU Radius Format String Vulnerability”,

http://bugs.gentoo.org/show_bug.cgi?id=156376, Nov. 26th, 2006

[32]. OpenRADIUS, "OpenRADIUS introduction", http://www.xs4all.nl/~evbergen/openradius/openradius-introduction.html, Mar. 25th, 2007

[33]. FreeRADIUS Wiki, "Other RADIUS Server", http://wiki.freeradius.org/Other_RADIUS_Servers#GNU_RADIUS, Jun. 21st, 2007

[34]. "Cistron RADIUS server, version 1.6.8", http://www.radius.cistron.nl/, Feb. 8th, 2006

[35]. Wikipedia, "BSDRadius", http://en.wikipedia.org/wiki/BSDRadius, May 15th. 2008

[36]. TeKRADIUS, design by Free CSS Templates, http://www.tekradius.com/, May 15th, 2008

[37]. Dhivakaran Muruganantham, Michael Helm, Tony Genovese, Roberto Morelli, and John Webster; "ESnet RAF Progress report", http://www.es.net/raf/ESnet%20RAF%20Progress%20Report%20Apr%202005.doc, May 26th, 2005

[38]. Market Wire, "New Version of Infoblox RADIUSone(TM) Appliance Delivers Enhanced AAA Services; Simplifies Wireless Deployments", http://findarticles.com/p/articles/mi_pwwi/is_200501/ai_n9477819, Jan., 2005

[39]. Business Wire, "Identity Engines Ignites Nationwide Channel With Marketlink; Marketlink Technologies Rapidly Achieves Outstanding Channel Traction With Identity Engines' Ignition™ Platform", http://findarticles.com/p/articles/mi_m0EIN/is_2006_April_10/ai_n16120467, Apr. 10th, 2006

[40]. idEngines, "Ignition Server-Precision Made Simple", http://www.idengines.com/products/ignitionserver/, Aug. 25th, 2008

[41]. Pallette Multimedia Pte Ltd, "computing & IT services", http://sg.88db.com/sg/Services/Post_Detail.page/business_services/it_services/?PostID=69495, Dec. 1st, 2006

[42]. Wikipedia, "List of RADIUS Servers", http://en.wikipedia.org/wiki/List_of_RADIUS_Servers, May 17th, 2008

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

[44]. Cisco, "Authentication, authorization, and Accounting overview- Differences between RADIUS and DIAMETER" , http://www.cisco.com/en/US/products/ps6638/products_data_sheet09186a00804fe332.html, Jun. 20th, 2008

[45]. Arlinx, “freeRADIUS AAA Server”, http://www.arlinx.com/freeradius.html, Jun. 16th, 2008

[46]. freeRADIUS, “Press Peleases”, http://freeradius.org/press/index.html#2.0.5, Jun. 7th, 2008

Appendix I. The FreeRADIUS sever set up process

This thesis is working under the Linux system: openSUSE. And the processes introduced here are based on this system.

1. Install the FreeRADIUS and other support resource

The openSUSE system has "freeradius" in its install packet. This FreeRADIUS may not be the latest version, but it is steadily combined with the system. Other support resource like openssl, mysql can also be found in the openSUSE install packet and recommended. All the installations can be finished by go to: "YaST -> System -> System Services (Runlevel)".

If the FreeRADIUS which provide in the system is not new enough, the server manager can install the latest version by download the install packet from the FreeRADIUS home website: http://freeradius.org/. Then the FreeRADIUS can be installed by following steps:

1). tar zxvf ./freeradius-server-2.0.5.tar.gz (decompressing files)

2). cd freeradius-server-2.0.5 (go to the file of FreeRADIUS)

3). ./configure

4). make

5). make install (run this command as root user)

2. Configure the FreeRADIUS connecting with MySQL

This step will build a MySQL database for the FreeRADIUS. The first step is creating a MySQL database for the RADIUS server. The system manager should define a root user password when the SQL database created. The command "mysqladmin -uroot password <yourpassword>" can achieved this when root use the SQL database for the first time. Then a "radius" database can be created.

```
mysql -uroot -p <password>
mysql> create database radius;
mysql>\q
```

The configuration would be made to make the FreeRADIUS using MySQL. Files under: /usr/local/etc/raddb would be modified as shown below:

--radiusd.conf:

```
        authorize {
        ……
        sql（removed #）
        ……
        }
        accouting {
        ….
        sql （removed#）
        …
        }
--sql.conf
        server=”localhost”
        login=”root”
        password=” <mysql root password>”
        radius_db=”radius”
--clients.conf
```

This part has contains the local testing information, the NAS connecting secret is “testing123”, this part can be modified.

3. Run the FreeRADIUS server

After the FreeRADIUS has install and configured running MySQL, the server manager can go to the /usr/local/sbin to start the RADIUS server. “radiusd or” is used to start RADIUS server, and “radiusd –X” is used to start RADIUS server in debug mode.

In the debug mode, if the message comes out as:

```
Listening on authentication address * port 1812
Listening on accounting address * port 1813
Listening on proxy address * port 1814
Ready to process requests.
```

That means the FreeRADIUS is running well.

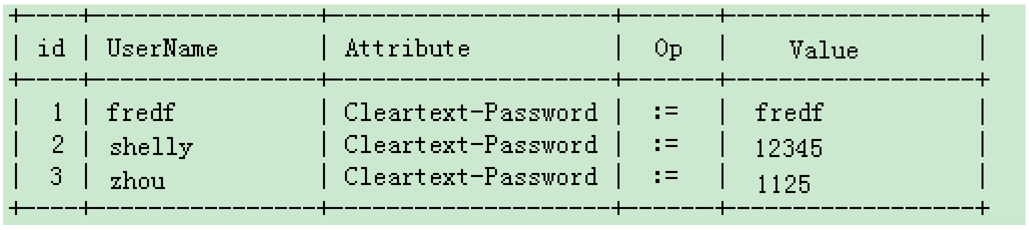
4. Add new information in MySQL database.

Different information can be added in the “radius” MySQL database by following command:

mysql -uroot -p <password> (enter to the MySQL database)

mysql> select * from radcheck;

(This is used to check the user information in radcheck, which normal contains the user name and password information. The information will be listed as showed below.)



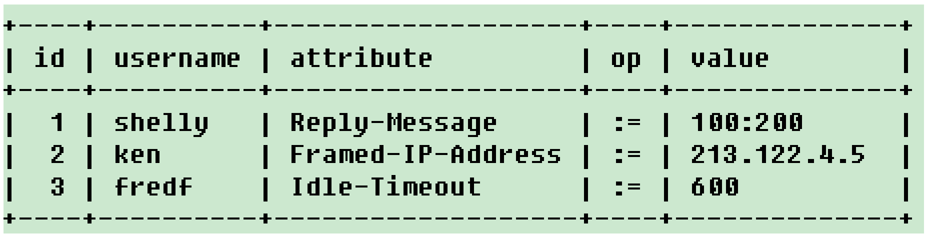
```
+----+----------------+--------------------+-------+------------------+
| id | UserName       | Attribute          |  Op   |     Value        |
+----+----------------+--------------------+-------+------------------+
|  1 | fredf          | Cleartext-Password |  :=   |  fredf           |
|  2 | shelly         | Cleartext-Password |  :=   |  12345           |
|  3 | zhou           | Cleartext-Password |  :=   |  1125            |
+----+----------------+--------------------+-------+------------------+
```

mysql> insert into radcheck values(4, “jia”, “Cleartext-Password”, “:=”, “1234”);

(This command is used to add a user named “jia” and the password is “1234”.)

mysql> select * from radreply;

(This is used to check the information in “radreply” file.)



```
+----+----------+-------------------+----+---------------+
| id | username | attribute         | op | value         |
+----+----------+-------------------+----+---------------+
|  1 | shelly   | Reply-Message     | := | 100:200       |
|  2 | ken      | Framed-IP-Address | := | 213.122.4.5   |
|  3 | fredf    | Idle-Timeout      | := | 600           |
+----+----------+-------------------+----+---------------+
```

mysql> insert into radreply values(4, “jia”, “KTH-default-bandwidth”, “:=”, “120”);

(This command is used to add a KTH-default-bandwidth value 120 to the database.)

mysql>\q

Appendix II. New codes for FreeRADIUS

1. Adding global attributes for bandwidth information

Occupy Values 192-223 which are reserved for experimental use to add new attributes.

a) in radius.h add

```
/* here we define our own attributes, which are
*   Default-Bandwidth and Max-Bandwith.
*   These two attributes have been added for non-binary
*   authentication experiment
*/
#define PW_DEFAULT_BANDWIDTH            193
#define PW_MAX_BANDWIDTH                194
```

b) in /etc/raddb/dictionary add

```
# New Attributes are defined here.
ATTRIBUTE        Default-BandWidth        193        string
ATTRIBUTE        Max-Bandwidth            194        string
```

c) include /etc/raddb/dictionary to /usr/local/etc/raddb/dictionary

```
$INCLUDE            /etc/raddb/dictionary
```

Then, when sql module queries the radreply table, it could search the dictionary file and convert text-based attributes to corresponding integer number.

2. Adding vendor specific attributes for bandwidth information

a) create /usr/local/share/freeradius/dictionary.kth, add:

```
    # -*- text -*-
##############################################################################
#
#    KTH dictionary
#
# 2008.09.01 GQMJr for experimental use
#
##############################################################################
VENDOR        KTH           933
BEGIN-VENDOR    KTH
ATTRIBUTE    KTH-default-bandwidth        3     integer
ATTRIBUTE    KTH-max-bandwidth            4     integer
```

```
END-VENDOR    KTH
```

b) include it to /usr/local/share/freeradius/dictionary

```
    $INCLUDE dictionary.kth
```

3. Modification of the radclient.c

```
// modified part of radclient.c

            /* handle DEFAULT bandwidth parameters */
            if (reply->code !=PW_AUTHENTICATION_REJECT) {
                        VALUE_PAIR *vp;

                        /* Find the pair with attribute PW_DEFAULT_BANDWIDTH*/
                        vp = reply->vps;

                        while(vp && vp->attribute != PW_DEFAULT_BANDWIDTH)
                                    vp = vp->next;

                        if( vp == NULL ) {
                                    printf("The packet received does not contain
DEFAULT bandwith information\n");
                                    goto packet_done;
                        }
                        else {
                        /* Extract the bandwidth information from received packet.
                          */
                                    int    default_bandwidth;
                                    char buf[128];
                              /* Here 128 was used to make the buffer big enough, but not
to much redundant.*/

                                    strcpy(buf, vp->vp_strvalue);
                                    default_bandwidth = atoi(buf);

                                    printf("The DEFAULT bandwidth information recevied
is:\n");
                                    printf("default_bandwidth: %d\n", default_bandwidth);

                        }
            }

            /* handle MAX bandwidth parameters */
            if (reply->code !=PW_AUTHENTICATION_REJECT) {
                VALUE_PAIR *vp;

                /* Find the pair with attribute PW_MAX_BANDWIDTH*/
                vp = reply->vps;

                while(vp && vp->attribute != PW_MAX_BANDWIDTH)
                        vp = vp->next;

                if( vp == NULL ) {
                        printf("The packet received does not contain MAX
bandwith information\n");
                        goto packet_done;
                }
                else {
                /* Extract the bandwidth information from received packet.
                 */
                        int    max_bandwidth;
                        char buf[128];

                        strcpy(buf, vp->vp_strvalue);
                        max_bandwidth = atoi(buf);
                        printf("The MAX bandwidth information received
is:\n");
                        printf("max_bandwidth: %d\n", max_bandwidth);

                }
        }
```

4. Modification of eap.c

```
/*
         *      EAP-Message is always associated with Message-Authenticator but not
vice-versa.
         *
         *      Don't add a Message-Authenticator if it's already there.
         */
        vp = pairfind(request->reply->vps, PW_MESSAGE_AUTHENTICATOR);
        if (!vp) {
                vp = paircreate(PW_MESSAGE_AUTHENTICATOR, PW_TYPE_OCTETS);
                memset(vp->vp_octets, 0, AUTH_VECTOR_LEN);
                vp->length = AUTH_VECTOR_LEN;

                pairadd(&(request->reply->vps), vp);
        }
        /* only output the reply attribute when reply code is PW_EAP_SUCCESS */
        if ((!request->reply->code) && (reply->code != PW_EAP_SUCCESS)) {
                pairdelete(&request->reply->vps, PW_DEFAULT_BANDWIDTH);
                pairdelete(&request->reply->vps, PW_MAX_BANDWIDTH);
                pairdelete(&request->reply->vps, PW_KTH_Default_Bandwidth);
                pairdelete(&request->reply->vps, PW_KTH_Max_Bandwidth);
                vp_printlist(stdout, request->reply->vps);
        }
```

5. Bandwidth negotiation function code

```
/the bandwidth negotiation function

int band_width (int user_name int request_width)
{
    int max_width;
    max_width = value_sql (user_name, attribute value=194);

    if (request_width > max_width)
    {
        return 0;
    }
    else
    {
        return request_width;
    }

}
```

Appendix III. All test results in this thesis

1. Authentication success results (without any new attributes)

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main # ./radtest
        shelly hello localhost 1812 testing123
Sending Access-Request of id 140 to 127.0.0.1 port 1812
         User-Name = "shelly"
         User-Password = "hello"
         NAS-IP-Address = 127.0.0.2
         NAS-Port = 1812

rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=140,
        length=29
         Reply-Message = "100:200"

The bandwidth information recevied is:
default_bandwidth: 100
max_bandwidth: 200
```

2. Authentication success results (with new attributes) by "radtest" command

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main      ./radtest
          fredf fredf localhost 1812 testing123
Sending Access-Request of id 214 to 127.0.0.1 port 1812
          User-Name = "fredf"
          User-Password = "fredf"
          NAS-IP-Address = 127.0.0.2
          NAS-Port = 1812

rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=214,
          length=68
          Reply-Message = "Hello fredf!"
          Default-BandWidth = "100"
          Max-Bandwidth = "200"
          KTH-default-bandwidth = 3
          KTH-max-bandwidth = 4

Received response ID 214, code 2, length = 68
          Reply-Message = "Hello fredf!"
          Default-BandWidth = "100"
          Max-Bandwidth = "200"
          KTH-default-bandwidth = 3
          KTH-max-bandwidth = 4
          name: Reply-Message       attribute: 18     vendor: 0
          name: Default-BandWidth attribute: 193    vendor: 0
          name: Max-Bandwidth       attribute: 194    vendor: 0
          vendor: KTH    name: KTH-default-bandwidth    attribute: 3    vendor: 933
          vendor: KTH    name: KTH-max-bandwidth      attribute: 4       vendor: 933

    The DEFAULT bandwidth information recevied is:
    default_bandwidth: 100
    The MAX bandwidth information received is:
    max_bandwidth: 200
```

3. Authentication success results (with new attributes) by “eaptest” command

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main # ./eaptest
            fred fred
+++> About to send encoded packet:
    User-Name = "fredf"
    Cleartext-Password = "fredf"
    EAP-Code = Response
    EAP-Id = 210
    EAP-Type-Identity = "fredf"
    Message-Authenticator = 0x00

<+++ EAP decoded packet:
    Reply-Message = "Hello fredf!"
    EAP-Message = 0x01d3001604102157l6a8606857fd387004c142548c70
    Message-Authenticator = 0x7efaaa66b147e2b35ad0a6c21ff61135
    State = 0x05de8aa1050d8ebbd18207d5bd212ccc
    EAP-Id = 211
    EAP-Code = Request
    EAP-Type-MD5 = 0x10215716a8606857fd387004c142548c70

+++> About to send encoded packet:
    User-Name = "fredf"
    Cleartext-Password = "fredf"
    EAP-Code = Response
    EAP-Id = 211
    Message-Authenticator = 0x00000000000000000000000000000000
    EAP-Type-MD5 = 0x102cd41af3a1c18cc935aea4ef7274f1d6
    State = 0x05de8aa1050d8ebbd18207d5bd212ccc

<+++ EAP decoded packet:
    Reply-Message = "Hello fredf!"
    Default-BandWidth = "100"
    Max-Bandwidth = "200"
    KTH-default-bandwidth = 3
    KTH-max-bandwidth = 4
    EAP-Message = 0x03d30004
    Message-Authenticator = 0xc3a8dfc9d623f39f293137a5d98e6fdc
    User-Name = "fredf"
    EAP-Id = 211
    EAP-Code = Success
```

4. Authentication results when password is incorrect by "radtest" command

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main # ./radtest
            fredf nikos localhost 1812 testing123
Sending Access-Request of id 26 to 127.0.0.1 port 1812
    User-Name = "fredf"
    User-Password = "nikos"
    NAS-IP-Address = 127.0.0.2
    NAS-Port = 1812

rad_recv: Access-Reject packet from host 127.0.0.1 port 1812, id=26,
            length=34
     Reply-Message = "Hello fredf!"
     Received response ID 26, code 3, length = 34
     Reply-Message = "Hello fredf!"
     name: Reply-Message      attribute: 18    vendor: 0

The packet received does not contain DEFAULT bandwith information
The packet received does not contain MAX bandwith information
```

5. Authentication results when password is incorrect by "eaptest" command

```
ccscenter:/home/jia/Desktop/jia/freeradius-server-2.0.5/src/main # ./eaptest
            jia 12345
+++> About to send encoded packet:
    User-Name = "jia"
     Cleartext-Password = "12345"
     EAP-Code = Response
     EAP-Id = 210
     EAP-Type-Identity = "jia"
     Message-Authenticator = 0x00

<+++ EAP decoded packet:
     EAP-Message = 0x01d30016041050d5a5903e3909bbbd8b243f0e32c046
     Message-Authenticator = 0x7d2be2347c0665e73cfdf98964c41584
     State = 0xc17a5dbdc1a95900978f167a5e24092b
     EAP-Id = 211
     EAP-Code = Request
     EAP-Type-MD5 = 0x1050d5a5903e3909bbbd8b243f0e32c046

+++> About to send encoded packet:
     User-Name = "jia"
     Cleartext-Password = "12345"
     EAP-Code = Response
     EAP-Id = 211
     Message-Authenticator = 0x00000000000000000000000000000000
     EAP-Type-MD5 = 0x1085a7272f27adac8d71c8d1daf6995136
     State = 0xc17a5dbdc1a95900978f167a5e24092b

<+++ EAP decoded packet:
     EAP-Message = 0x04d30004
     Message-Authenticator = 0x0ebf5331dc6aed18ef9cfa86610fdaa9
     EAP-Id = 211
     EAP-Code = Failure
```

Appendix IV. The working process log for Non-binary RADIUS when authentication succeeds

```
FreeRADIUS Version 2.0.5, for host i686-pc-linux-gnu, built on Sep  2 2008 at 13:26:46
Copyright (C) 1999-2008 The FreeRADIUS server project and contributors.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
You may redistribute copies of FreeRADIUS under the terms of the
GNU General Public License v2.
Starting - reading configuration files ...
including configuration file /usr/local/etc/raddb/radiusd.conf
including configuration file /usr/local/etc/raddb/proxy.conf
including configuration file /usr/local/etc/raddb/clients.conf
including configuration file /usr/local/etc/raddb/snmp.conf
including files in directory /usr/local/etc/raddb/modules/
including configuration file /usr/local/etc/raddb/modules/expiration
including configuration file /usr/local/etc/raddb/modules/smbpasswd
including configuration file /usr/local/etc/raddb/modules/attr_rewrite
including configuration file /usr/local/etc/raddb/modules/echo
including configuration file /usr/local/etc/raddb/modules/mac2ip
including configuration file /usr/local/etc/raddb/modules/attr_filter
including configuration file /usr/local/etc/raddb/modules/ldap
including configuration file /usr/local/etc/raddb/modules/krb5
including configuration file /usr/local/etc/raddb/modules/sradutmp
including configuration file /usr/local/etc/raddb/modules/passwd
including configuration file /usr/local/etc/raddb/modules/acct_unique
including configuration file /usr/local/etc/raddb/modules/logintime
including configuration file /usr/local/etc/raddb/modules/counter
including configuration file /usr/local/etc/raddb/modules/chap
including configuration file /usr/local/etc/raddb/modules/files
including configuration file /usr/local/etc/raddb/modules/exec
including configuration file /usr/local/etc/raddb/modules/digest
including configuration file /usr/local/etc/raddb/modules/mac2vlan
including configuration file /usr/local/etc/raddb/modules/sql_log
including configuration file /usr/local/etc/raddb/modules/pam
including configuration file /usr/local/etc/raddb/modules/realm
including configuration file /usr/local/etc/raddb/modules/pap
including configuration file /usr/local/etc/raddb/modules/mschap
including configuration file /usr/local/etc/raddb/modules/checkval
including configuration file /usr/local/etc/raddb/modules/expr
including configuration file /usr/local/etc/raddb/modules/etc_group
including configuration file /usr/local/etc/raddb/modules/ippool
including configuration file /usr/local/etc/raddb/modules/radutmp
including configuration file /usr/local/etc/raddb/modules/detail.log
including configuration file /usr/local/etc/raddb/modules/preprocess
including configuration file /usr/local/etc/raddb/modules/unix
including configuration file /usr/local/etc/raddb/modules/always
including configuration file /usr/local/etc/raddb/modules/detail
including configuration file /usr/local/etc/raddb/modules/policy
including configuration file /usr/local/etc/raddb/eap.conf
including configuration file /usr/local/etc/raddb/sql.conf
including configuration file /usr/local/etc/raddb/sql/MySQL/dialup.conf
including configuration file /usr/local/etc/raddb/sql/MySQL/counter.conf
including configuration file /usr/local/etc/raddb/policy.conf
including files in directory /usr/local/etc/raddb/sites-enabled/
including configuration file /usr/local/etc/raddb/sites-enabled/inner-tunnel
including configuration file /usr/local/etc/raddb/sites-enabled/default
including dictionary file /usr/local/etc/raddb/dictionary
main {
    prefix = "/usr/local"
    localstatedir = "/usr/local/var"
    logdir = "/usr/local/var/log/radius"
    libdir = "/usr/local/lib"
    radacctdir = "/usr/local/var/log/radius/radacct"
    hostname_lookups = no
    max_request_time = 30
    cleanup_delay = 5
    max_requests = 1024
    allow_core_dumps = no
    pidfile = "/usr/local/var/run/radiusd/radiusd.pid"
    checkrad = "/usr/local/sbin/checkrad"
    debug_level = 0
    proxy_requests = yes
 log {
    stripped_names = no
    auth = no
    auth_badpass = no
    auth_goodpass = no
 }
}
 client localhost {
    ipaddr = 127.0.0.1
    require_message_authenticator = no
    secret = "testing123"
    nastype = "other"
 }
```

```
radiusd: #### Loading Realms and Home Servers ####
 proxy server {
    retry_delay = 5
    retry_count = 3
    default_fallback = no
    dead_time = 120
    wake_all_if_all_dead = no
 }
 home_server localhost {
    ipaddr = 127.0.0.1
    port = 1812
    type = "auth"
    secret = "testing123"
    response_window = 20
    max_outstanding = 65536
    zombie_period = 40
    status_check = "status-server"
    ping_check = "none"
    ping_interval = 30
    check_interval = 30
    num_answers_to_alive = 3
    num_pings_to_alive = 3
    revive_interval = 120
    status_check_timeout = 4
 }
 home_server_pool my_auth_failover {
    type = fail-over
    home_server = localhost
 }
 realm example.com {
    auth_pool = my_auth_failover
 }
 realm LOCAL {
 }
radiusd: #### Instantiating modules ####
 instantiate {
 Module: Linked to module rlm_exec
 Module: Instantiating exec
  exec {
    wait = no
    input_pairs = "request"
    shell_escape = yes
  }
 Module: Linked to module rlm_expr
 Module: Instantiating expr
 Module: Linked to module rlm_expiration
 Module: Instantiating expiration
  expiration {
    reply-message = "Password Has Expired  "
  }
 Module: Linked to module rlm_logintime
 Module: Instantiating logintime
  logintime {
    reply-message = "You are calling outside your allowed timespan  "
    minimum-timeout = 60
  }
 }
radiusd: #### Loading Virtual Servers ####
server inner-tunnel {
 modules {
 Module: Checking authenticate {...} for more modules to load
 Module: Linked to module rlm_pap
 Module: Instantiating pap
  pap {
    encryption_scheme = "auto"
    auto_header = no
  }
 Module: Linked to module rlm_chap
 Module: Instantiating chap
 Module: Linked to module rlm_mschap
 Module: Instantiating mschap
  mschap {
    use_mppe = yes
    require_encryption = no
    require_strong = no
    with_ntdomain_hack = no
  }
 Module: Linked to module rlm_unix
 Module: Instantiating unix
  unix {
    radwtmp = "/usr/local/var/log/radius/radwtmp"
  }
 Module: Linked to module rlm_eap
 Module: Instantiating eap
  eap {
    default_eap_type = "md5"
    timer_expire = 60
    ignore_unknown_eap_types = no
```

```
   cisco_accounting_username_bug = no
  }
Module: Linked to sub-module rlm_eap_md5
Module: Instantiating eap-md5
Module: Linked to sub-module rlm_eap_leap
Module: Instantiating eap-leap
Module: Linked to sub-module rlm_eap_gtc
Module: Instantiating eap-gtc
  gtc {
   challenge = "Password: "
   auth_type = "PAP"
  }
Module: Linked to sub-module rlm_eap_tls
Module: Instantiating eap-tls
  tls {
   rsa_key_exchange = no
   dh_key_exchange = yes
   rsa_key_length = 512
   dh_key_length = 512
   verify_depth = 0
   pem_file_type = yes
   private_key_file = "/usr/local/etc/raddb/certs/server.pem"
   certificate_file = "/usr/local/etc/raddb/certs/server.pem"
   CA_file = "/usr/local/etc/raddb/certs/ca.pem"
   private_key_password = "whatever"
   dh_file = "/usr/local/etc/raddb/certs/dh"
   random_file = "/usr/local/etc/raddb/certs/random"
   fragment_size = 1024
   include_length = yes
   check_crl = no
   cipher_list = "DEFAULT"
   make_cert_command = "/usr/local/etc/raddb/certs/bootstrap"
  }
Module: Linked to sub-module rlm_eap_ttls
Module: Instantiating eap-ttls
  ttls {
   default_eap_type = "md5"
   copy_request_to_tunnel = no
   use_tunneled_reply = no
   virtual_server = "inner-tunnel"
  }
Module: Linked to sub-module rlm_eap_peap
Module: Instantiating eap-peap
  peap {
   default_eap_type = "mschapv2"
   copy_request_to_tunnel = no
   use_tunneled_reply = no
   proxy_tunneled_request_as_eap = yes
   virtual_server = "inner-tunnel"
  }
Module: Linked to sub-module rlm_eap_mschapv2
Module: Instantiating eap-mschapv2
  mschapv2 {
   with_ntdomain_hack = no
  }
Module: Checking authorize {...} for more modules to load
Module: Linked to module rlm_realm
Module: Instantiating suffix
 realm suffix {
   format = "suffix"
   delimiter = "@"
   ignore_default = no
   ignore_null = no
 }
Module: Linked to module rlm_files
Module: Instantiating files
 files {
   usersfile = "/usr/local/etc/raddb/users"
   acctusersfile = "/usr/local/etc/raddb/acct_users"
   preproxy_usersfile = "/usr/local/etc/raddb/preproxy_users"
   compat = "no"
 }
Module: Checking session {...} for more modules to load
Module: Linked to module rlm_radutmp
Module: Instantiating radutmp
 radutmp {
   filename = "/usr/local/var/log/radius/radutmp"
   username = "%{User-Name}"
   case_sensitive = yes
   check_with_nas = yes
   perm = 384
   callerid = yes
 }
Module: Checking post-proxy {...} for more modules to load
Module: Checking post-auth {...} for more modules to load
Module: Linked to module rlm_attr_filter
Module: Instantiating attr_filter.access_reject
 attr_filter attr_filter.access_reject {
   attrsfile = "/usr/local/etc/raddb/attrs.access_reject"
   key = "%{User-Name}"
 }
}
}
server {
modules {
Module: Checking authenticate {...} for more modules to load
Module: Checking authorize {...} for more modules to load
Module: Linked to module rlm_preprocess
Module: Instantiating preprocess
 preprocess {
   huntgroups = "/usr/local/etc/raddb/huntgroups"
   hints = "/usr/local/etc/raddb/hints"
   with_ascend_hack = no
   ascend_channels_per_line = 23
   with_ntdomain_hack = no
   with_specialix_jetstream_hack = no
   with_cisco_vsa_hack = no
   with_alvarion_vsa_hack = no
 }
Module: Linked to module rlm_sql
Module: Instantiating sql
 sql {
   driver = "rlm_sql_MySQL"
   server = "localhost"
   port = ""
   login = "radius"
   password = "radpass"
   radius_db = "radius"
   read_groups = yes
   sqltrace = no
   sqltracefile = "/usr/local/var/log/radius/sqltrace.sql"
   readclients = no
   deletestalesessions = yes
   num_sql_socks = 5
   sql_user_name = "%{User-Name}"
   default_user_profile = ""
   nas_query = "SELECT id, nasname, shortname, type, secret FROM nas"
   authorize_check_query = "SELECT id, username, attribute, value, op           FROM
radcheck           WHERE username = '%{SQL-User-Name}'           ORDER BY id"
   authorize_reply_query = "SELECT id, username, attribute, value, op           FROM
radreply           WHERE username = '%{SQL-User-Name}'           ORDER BY id"
   authorize_group_check_query = "SELECT id, groupname, attribute,           Value,
op          FROM radgroupcheck          WHERE groupname = '%{Sql-Group}'
ORDER BY id"
   authorize_group_reply_query = "SELECT id, groupname, attribute,          value,
op          FROM radgroupreply          WHERE groupname = '%{Sql-Group}'
ORDER BY id"
   accounting_onoff_query = "          UPDATE radacct          SET
acctstoptime      =  '%S',            acctsessiontime    =  unix_timestamp('%S')
-                                    unix_timestamp(acctstarttime),
acctterminatecause =  '%{Acct-Terminate-Cause}',            acctstopdelay      =
%{%{Acct-Delay-Time}:-0}          WHERE acctstoptime IS NULL          AND
nasipaddress      =  '%{NAS-IP-Address}'          AND acctstarttime     <= '%S'"
   accounting_update_query = "          UPDATE radacct          SET
framedipaddress = '%{Framed-IP-Address}',          acctsessiontime     =
'%{Acct-Session-Time}',          acctinputoctets     =
'%{%{Acct-Input-Gigawords}:-0}'  << 32 |
'%{%{Acct-Input-Octets}:-0}',          acctoutputoctets    =
'%{%{Acct-Output-Gigawords}:-0}' << 32 |
'%{%{Acct-Output-Octets}:-0}'          WHERE acctsessionid = '%{Acct-Session-Id}'
AND username        = '%{SQL-User-Name}'          AND nasipaddress    =
'%{NAS-IP-Address}'"
   accounting_update_query_alt = "          INSERT INTO radacct
(acctsessionid,    acctuniqueid,      username,            realm,
nasipaddress,      nasportid,            nasporttype,      acctstarttime,
acctsessiontime,            acctauthentic,    connectinfo_start, acctinputoctets,
acctoutputoctets, calledstationid,   callingstationid,            servicetype,
framedprotocol,    framedipaddress,            acctstartdelay,
xascendsessionsvrkey)          VALUES            ('%{Acct-Session-Id}',
'%{Acct-Unique-Session-Id}',            '%{SQL-User-Name}',
'%{Realm}', '%{NAS-IP-Address}', '%{NAS-Port}',            '%{NAS-Port-Type}',
DATE_SUB('%S',                    INTERVAL (%{%{Acct-Session-Time}:-0} +
%{%{Acct-Delay-Time}:-0}) SECOND),                '%{Acct-Session-Time}',
'%{Acct-Authentic}', '',            '%{%{Acct-Input-Gigawords}:-0}' << 32 |
'%{%{Acct-Input-Octets}:-0}',            '%{%{Acct-Output-Gigawords}:-0}' << 32 |
'%{%{Acct-Output-Octets}:-0}',            '%{Called-Station-Id}',
'%{Calling-Station-Id}',            '%{Service-Type}', '%{Framed-Protocol}',
'%{Framed-IP-Address}',            '0', '%{X-Ascend-Session-Svr-Key}')"
   accounting_start_query = "          INSERT INTO radacct
(acctsessionid,    acctuniqueid,     username,            realm,
nasipaddress,     nasportid,            nasporttype,      acctstarttime,
acctstoptime,            acctsessiontime,  acctauthentic,    connectinfo_start,
connectinfo_stop, acctinputoctets,  acctoutputoctets,            calledstationid,
callingstationid, acctterminatecause,            servicetype,       framedprotocol,
framedipaddress,            acctstartdelay,   acctstopdelay,
xascendsessionsvrkey)          VALUES            ('%{Acct-Session-Id}',
'%{Acct-Unique-Session-Id}',              '%{SQL-User-Name}',
'%{Realm}', '%{NAS-IP-Address}', '%{NAS-Port}',              '%{NAS-Port-Type}',
'%S', NULL,              '0', '%{Acct-Authentic}', '%{Connect-Info}',
'', '0', '0',              '%{Called-Station-Id}', '%{Calling-Station-Id}', '',
'%{Service-Type}', '%{Framed-Protocol}', '%{Framed-IP-Address}',
'%{%{Acct-Delay-Time}:-0}', '0', '%{X-Ascend-Session-Svr-Key}')"
    accounting_start_query_alt = "           UPDATE radacct SET
acctstarttime     = '%S',              acctstartdelay    =
'%{%{Acct-Delay-Time}:-0}',              connectinfo_start = '%{Connect-Info}'
WHERE acctsessionid  = '%{Acct-Session-Id}'           AND username         =
'%{SQL-User-Name}'            AND nasipaddress     = '%{NAS-IP-Address}'"
    accounting_stop_query = "           UPDATE radacct SET              acctstoptime
= '%S',              acctsessiontime    = '%{Acct-Session-Time}',
acctinputoctets    = '%{%{Acct-Input-Gigawords}:-0}' << 32 |
'%{%{Acct-Input-Octets}:-0}',              acctoutputoctets   =
'%{%{Acct-Output-Gigawords}:-0}' << 32 |
'%{%{Acct-Output-Octets}:-0}',              acctterminatecause =
'%{Acct-Terminate-Cause}',              acctstopdelay      =
'%{%{Acct-Delay-Time}:-0}',              connectinfo_stop   = '%{Connect-Info}'
WHERE acctsessionid   = '%{Acct-Session-Id}'           AND username          =
'%{SQL-User-Name}'           AND nasipaddress      = '%{NAS-IP-Address}'"
    accounting_stop_query_alt = "           INSERT INTO radacct
(acctsessionid, acctuniqueid, username,              realm, nasipaddress, nasportid,
nasporttype, acctstarttime, acctstoptime,              acctsessiontime,
acctauthentic, connectinfo_start,              connectinfo_stop, acctinputoctets,
acctoutputoctets,              calledstationid, callingstationid, acctterminatecause,
servicetype, framedprotocol, framedipaddress,              acctstartdelay,
acctstopdelay)           VALUES           ('%{Acct-Session-Id}',
'%{Acct-Unique-Session-Id}',              '%{SQL-User-Name}',
'%{Realm}', '%{NAS-IP-Address}', '%{NAS-Port}',              '%{NAS-Port-Type}',
DATE_SUB('%S',              INTERVAL (%{%{Acct-Session-Time}:-0} +
%{%{Acct-Delay-Time}:-0}) SECOND),              '%S', '%{Acct-Session-Time}',
'%{Acct-Authentic}', '',              '%{Connect-Info}',
'%{%{Acct-Input-Gigawords}:-0}' << 32 |              '%{%{Acct-Input-Octets}:-0}',
'%{%{Acct-Output-Gigawords}:-0}' << 32 |              '%{%{Acct-Output-Octets}:-0}',
'%{Called-Station-Id}', '%{Calling-Station-Id}',
'%{Acct-Terminate-Cause}',              '%{Service-Type}', '%{Framed-Protocol}',
'%{Framed-IP-Address}',              '0', '%{%{Acct-Delay-Time}:-0}')"
    group_membership_query = "SELECT groupname           FROM radusergroup
WHERE username = '%{SQL-User-Name}'           ORDER BY priority"
    connect_failure_retry_delay = 60
    simul_count_query = ""
    simul_verify_query = "SELECT radacctid, acctsessionid, username,
nasipaddress, nasportid, framedipaddress,
callingstationid, framedprotocol                                FROM radacct
WHERE username = '%{SQL-User-Name}'                              AND acctstoptime
IS NULL"
    postauth_query = "INSERT INTO radpostauth                          (username,
pass, reply, authdate)                            VALUES
(                            '%{User-Name}',
'%{%{User-Password}:-%{Chap-Password}}',
'%{reply:Packet-Type}', '%S')"
    safe-characters =
"@abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.-_: /"
  }
rlm_sql (sql): Driver rlm_sql_MySQL (module rlm_sql_MySQL) loaded and linked
rlm_sql (sql): Attempting to connect to radius@localhost:/radius
rlm_sql (sql): starting 0
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #0
rlm_sql_MySQL: Starting connect to MySQL server for #0
rlm_sql (sql): Connected new DB handle, #0
rlm_sql (sql): starting 1
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #1
rlm_sql_MySQL: Starting connect to MySQL server for #1
rlm_sql (sql): Connected new DB handle, #1
rlm_sql (sql): starting 2
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #2
rlm_sql_MySQL: Starting connect to MySQL server for #2
rlm_sql (sql): Connected new DB handle, #2
rlm_sql (sql): starting 3
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #3
rlm_sql_MySQL: Starting connect to MySQL server for #3
rlm_sql (sql): Connected new DB handle, #3
rlm_sql (sql): starting 4
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #4
rlm_sql_MySQL: Starting connect to MySQL server for #4
rlm_sql (sql): Connected new DB handle, #4
 Module: Checking preacct {...} for more modules to load
 Module: Linked to module rlm_acct_unique
 Module: Instantiating acct_unique
  acct_unique {
    key = "User-Name, Acct-Session-Id, NAS-IP-Address, Client-IP-Address, NAS-Port"
  }
 Module: Checking accounting {...} for more modules to load
 Module: Linked to module rlm_detail
 Module: Instantiating detail
  detail {
    detailfile =
"/usr/local/var/log/radius/radacct/%{Client-IP-Address}/detail-%Y%m%d"
    header = "%t"
    detailperm = 384
    dirperm = 493
    locking = no
    log_packet_header = no
  }
 Module: Instantiating attr_filter.accounting_response
  attr_filter attr_filter.accounting_response {
    attrsfile = "/usr/local/etc/raddb/attrs.accounting_response"
    key = "%{User-Name}"
  }
 Module: Checking session {...} for more modules to load
 Module: Checking post-proxy {...} for more modules to load
 Module: Checking post-auth {...} for more modules to load
 }
}
radiusd: #### Opening IP addresses and Ports ####
listen {
    type = "auth"
    ipaddr = *
    port = 0
}
listen {
    type = "acct"
    ipaddr = *
    port = 0
}
Listening on authentication address * port 1812
Listening on accounting address * port 1813
Listening on proxy address * port 1814
Ready to process requests.
rad_recv: Access-Request packet from host 127.0.0.1 port 1029, id=214, length=57
    User-Name = "fredf"
    User-Password = "fredf"
    NAS-IP-Address = 127.0.0.2
    NAS-Port = 1812
+- entering group authorize
++[preprocess] returns ok
++[chap] returns noop
++[mschap] returns noop
    rlm_realm: No '@' in User-Name = "fredf", looking up realm NULL
    rlm_realm: No such realm "NULL"
++[suffix] returns noop
```

```
  rlm_eap: No EAP-Message, not doing EAP
++[eap] returns noop
++[unix] returns notfound
++[files] returns noop
    expand: %{User-Name} -> fredf
rlm_sql (sql): sql_set_user escaped user --> 'fredf'
rlm_sql (sql): Reserving sql socket id: 4
    expand: SELECT id, username, attribute, value, op          FROM radcheck
WHERE username = '%{SQL-User-Name}'          ORDER BY id -> SELECT id, username,
attribute, value, op          FROM radcheck          WHERE username = 'fredf'
ORDER BY id
rlm_sql (sql): User found in radcheck table
    expand: SELECT id, username, attribute, value, op          FROM radreply
WHERE username = '%{SQL-User-Name}'          ORDER BY id -> SELECT id, username,
attribute, value, op          FROM radreply          WHERE username = 'fredf'
ORDER BY id
    expand: SELECT groupname          FROM radusergroup          WHERE username =
'%{SQL-User-Name}'          ORDER BY priority -> SELECT groupname          FROM
radusergroup          WHERE username = 'fredf'          ORDER BY priority
rlm_sql (sql): Released sql socket id: 4
++[sql] returns ok
++[expiration] returns noop
++[logintime] returns noop
++[pap] returns updated
  rad_check_password:  Found Auth-Type
auth: type "PAP"
+- entering group PAP
rlm_pap: login attempt with password "fredf"
rlm_pap: Using clear text password "fredf"
rlm_pap: User authenticated successfully
++[pap] returns ok
+- entering group post-auth
++[exec] returns noop
Sending Access-Accept of id 214 to 127.0.0.1 port 1029
    Reply-Message := "Hello fredf!"
    Default-BandWidth := "100"
    Max-Bandwidth := "200"
    KTH-default-bandwidth := 3
    KTH-max-bandwidth := 4
Finished request 0.
Going to the next request
Waking up in 4.9 seconds.
```

Appendix V. The working process log for Non-binary RADIUS when authentication fails

```
FreeRADIUS Version 2.0.5, for host i686-pc-linux-gnu, built on Sep  2 2008 at 13:26:46
Copyright (C) 1999-2008 The FreeRADIUS server project and contributors.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
You may redistribute copies of FreeRADIUS under the terms of the GNU General Public
License v2.
Starting - reading configuration files ...
including configuration file /usr/local/etc/raddb/radiusd.conf
including configuration file /usr/local/etc/raddb/proxy.conf
including configuration file /usr/local/etc/raddb/clients.conf
including configuration file /usr/local/etc/raddb/snmp.conf
including files in directory /usr/local/etc/raddb/modules/
including configuration file /usr/local/etc/raddb/modules/expiration
including configuration file /usr/local/etc/raddb/modules/smbpasswd
including configuration file /usr/local/etc/raddb/modules/attr_rewrite
including configuration file /usr/local/etc/raddb/modules/echo
including configuration file /usr/local/etc/raddb/modules/mac2ip
including configuration file /usr/local/etc/raddb/modules/attr_filter
including configuration file /usr/local/etc/raddb/modules/ldap
including configuration file /usr/local/etc/raddb/modules/krb5
including configuration file /usr/local/etc/raddb/modules/sradutmp
including configuration file /usr/local/etc/raddb/modules/passwd
including configuration file /usr/local/etc/raddb/modules/acct_unique
including configuration file /usr/local/etc/raddb/modules/logintime
including configuration file /usr/local/etc/raddb/modules/counter
including configuration file /usr/local/etc/raddb/modules/chap
including configuration file /usr/local/etc/raddb/modules/files
including configuration file /usr/local/etc/raddb/modules/exec
including configuration file /usr/local/etc/raddb/modules/digest
including configuration file /usr/local/etc/raddb/modules/mac2vlan
including configuration file /usr/local/etc/raddb/modules/sql_log
including configuration file /usr/local/etc/raddb/modules/pam
including configuration file /usr/local/etc/raddb/modules/realm
including configuration file /usr/local/etc/raddb/modules/pap
including configuration file /usr/local/etc/raddb/modules/mschap
including configuration file /usr/local/etc/raddb/modules/checkval
including configuration file /usr/local/etc/raddb/modules/expr
including configuration file /usr/local/etc/raddb/modules/etc_group
including configuration file /usr/local/etc/raddb/modules/ippool
including configuration file /usr/local/etc/raddb/modules/radutmp
including configuration file /usr/local/etc/raddb/modules/detail.log
including configuration file /usr/local/etc/raddb/modules/preprocess
including configuration file /usr/local/etc/raddb/modules/unix
including configuration file /usr/local/etc/raddb/modules/always
including configuration file /usr/local/etc/raddb/modules/detail
including configuration file /usr/local/etc/raddb/modules/policy
including configuration file /usr/local/etc/raddb/eap.conf
including configuration file /usr/local/etc/raddb/sql.conf
including configuration file /usr/local/etc/raddb/sql/MySQL/dialup.conf
including configuration file /usr/local/etc/raddb/sql/MySQL/counter.conf
including configuration file /usr/local/etc/raddb/policy.conf
including files in directory /usr/local/etc/raddb/sites-enabled/
including configuration file /usr/local/etc/raddb/sites-enabled/inner-tunnel
including configuration file /usr/local/etc/raddb/sites-enabled/default
including dictionary file /usr/local/etc/raddb/dictionary
main {
    prefix = "/usr/local"
    localstatedir = "/usr/local/var"
    logdir = "/usr/local/var/log/radius"
    libdir = "/usr/local/lib"
    radacctdir = "/usr/local/var/log/radius/radacct"
    hostname_lookups = no
    max_request_time = 30
    cleanup_delay = 5
    max_requests = 1024
    allow_core_dumps = no
    pidfile = "/usr/local/var/run/radiusd/radiusd.pid"
    checkrad = "/usr/local/sbin/checkrad"
    debug_level = 0
    proxy_requests = yes
 log {
    stripped_names = no
    auth = no
    auth_badpass = no
    auth_goodpass = no
 }
}
 client localhost {
    ipaddr = 127.0.0.1
    require_message_authenticator = no
    secret = "testing123"
    nastype = "other"
 }
radiusd: #### Loading Realms and Home Servers ####
```

```
 proxy server {
    retry_delay = 5
    retry_count = 3
    default_fallback = no
    dead_time = 120
    wake_all_if_all_dead = no
 }
 home_server localhost {
    ipaddr = 127.0.0.1
    port = 1812
    type = "auth"
    secret = "testing123"
    response_window = 20
    max_outstanding = 65536
    zombie_period = 40
    status_check = "status-server"
    ping_check = "none"
    ping_interval = 30
    check_interval = 30
    num_answers_to_alive = 3
    num_pings_to_alive = 3
    revive_interval = 120
    status_check_timeout = 4
 }
 home_server_pool my_auth_failover {
    type = fail-over
    home_server = localhost
 }
 realm example.com {
    auth_pool = my_auth_failover
 }
 realm LOCAL {
 }
radiusd: #### Instantiating modules ####
 instantiate {
 Module: Linked to module rlm_exec
 Module: Instantiating exec
  exec {
    wait = no
    input_pairs = "request"
    shell_escape = yes
  }
 Module: Linked to module rlm_expr
 Module: Instantiating expr
 Module: Linked to module rlm_expiration
 Module: Instantiating expiration
  expiration {
    reply-message = "Password Has Expired  "
  }
 Module: Linked to module rlm_logintime
 Module: Instantiating logintime
  logintime {
    reply-message = "You are calling outside your allowed timespan  "
    minimum-timeout = 60
  }
 }
radiusd: #### Loading Virtual Servers ####
server inner-tunnel {
 modules {
 Module: Checking authenticate {...} for more modules to load
 Module: Linked to module rlm_pap
 Module: Instantiating pap
  pap {
    encryption_scheme = "auto"
    auto_header = no
  }
 Module: Linked to module rlm_chap
 Module: Instantiating chap
 Module: Linked to module rlm_mschap
 Module: Instantiating mschap
  mschap {
    use_mppe = yes
    require_encryption = no
    require_strong = no
    with_ntdomain_hack = no
  }
 Module: Linked to module rlm_unix
 Module: Instantiating unix
  unix {
    radwtmp = "/usr/local/var/log/radius/radwtmp"
  }
 Module: Linked to module rlm_eap
 Module: Instantiating eap
  eap {
    default_eap_type = "md5"
    timer_expire = 60
    ignore_unknown_eap_types = no
    cisco_accounting_username_bug = no
  }
 Module: Linked to sub-module rlm_eap_md5
 Module: Instantiating eap-md5
 Module: Linked to sub-module rlm_eap_leap
 Module: Instantiating eap-leap
 Module: Linked to sub-module rlm_eap_gtc
 Module: Instantiating eap-gtc
   gtc {
    challenge = "Password: "
    auth_type = "PAP"
   }
 Module: Linked to sub-module rlm_eap_tls
 Module: Instantiating eap-tls
   tls {
    rsa_key_exchange = no
    dh_key_exchange = yes
    rsa_key_length = 512
    dh_key_length = 512
    verify_depth = 0
    pem_file_type = yes
    private_key_file = "/usr/local/etc/raddb/certs/server.pem"
    certificate_file = "/usr/local/etc/raddb/certs/server.pem"
    CA_file = "/usr/local/etc/raddb/certs/ca.pem"
    private_key_password = "whatever"
    dh_file = "/usr/local/etc/raddb/certs/dh"
    random_file = "/usr/local/etc/raddb/certs/random"
    fragment_size = 1024
    include_length = yes
    check_crl = no
    cipher_list = "DEFAULT"
    make_cert_command = "/usr/local/etc/raddb/certs/bootstrap"
   }
 Module: Linked to sub-module rlm_eap_ttls
 Module: Instantiating eap-ttls
   ttls {
    default_eap_type = "md5"
    copy_request_to_tunnel = no
    use_tunneled_reply = no
    virtual_server = "inner-tunnel"
   }
 Module: Linked to sub-module rlm_eap_peap
 Module: Instantiating eap-peap
   peap {
    default_eap_type = "mschapv2"
```

```
    copy_request_to_tunnel = no
    use_tunneled_reply = no
    proxy_tunneled_request_as_eap = yes
    virtual_server = "inner-tunnel"
  }
Module: Linked to sub-module rlm_eap_mschapv2
Module: Instantiating eap-mschapv2
  mschapv2 {
    with_ntdomain_hack = no
  }
Module: Checking authorize {...} for more modules to load
Module: Linked to module rlm_realm
Module: Instantiating suffix
 realm suffix {
    format = "suffix"
    delimiter = "@"
    ignore_default = no
    ignore_null = no
 }
Module: Linked to module rlm_files
Module: Instantiating files
 files {
    usersfile = "/usr/local/etc/raddb/users"
    acctusersfile = "/usr/local/etc/raddb/acct_users"
    preproxy_usersfile = "/usr/local/etc/raddb/preproxy_users"
    compat = "no"
 }
Module: Checking session {...} for more modules to load
Module: Linked to module rlm_radutmp
Module: Instantiating radutmp
 radutmp {
    filename = "/usr/local/var/log/radius/radutmp"
    username = "%{User-Name}"
    case_sensitive = yes
    check_with_nas = yes
    perm = 384
    callerid = yes
 }
Module: Checking post-proxy {...} for more modules to load
Module: Checking post-auth {...} for more modules to load
Module: Linked to module rlm_attr_filter
Module: Instantiating attr_filter.access_reject
 attr_filter attr_filter.access_reject {
    attrsfile = "/usr/local/etc/raddb/attrs.access_reject"
    key = "%{User-Name}"
  }
 }
}
server {
 modules {
 Module: Checking authenticate {...} for more modules to load
 Module: Checking authorize {...} for more modules to load
 Module: Linked to module rlm_preprocess
 Module: Instantiating preprocess
  preprocess {
    huntgroups = "/usr/local/etc/raddb/huntgroups"
    hints = "/usr/local/etc/raddb/hints"
    with_ascend_hack = no
    ascend_channels_per_line = 23
    with_ntdomain_hack = no
    with_specialix_jetstream_hack = no
    with_cisco_vsa_hack = no
    with_alvarion_vsa_hack = no
  }
 Module: Linked to module rlm_sql
 Module: Instantiating sql
  sql {
    driver = "rlm_sql_MySQL"
    server = "localhost"
    port = ""
    login = "radius"
    password = "radpass"
    radius_db = "radius"
    read_groups = yes
    sqltrace = no
    sqltracefile = "/usr/local/var/log/radius/sqltrace.sql"
    readclients = no
    deletestalesessions = yes
    num_sql_socks = 5
    sql_user_name = "%{User-Name}"
    default_user_profile = ""
    nas_query = "SELECT id, nasname, shortname, type, secret FROM nas"
    authorize_check_query = "SELECT id, username, attribute, value, op           FROM
radcheck           WHERE username = '%{SQL-User-Name}'           ORDER BY id"
    authorize_reply_query = "SELECT id, username, attribute, value, op           FROM
radreply           WHERE username = '%{SQL-User-Name}'           ORDER BY id"
    authorize_group_check_query = "SELECT id, groupname, attribute,           Value,
op           FROM radgroupcheck           WHERE groupname = '%{Sql-Group}'
```

```
ORDER BY id"
    authorize_group_reply_query = "SELECT id, groupname, attribute,           value,
op           FROM radgroupreply           WHERE groupname = '%{Sql-Group}'
ORDER BY id"
    accounting_onoff_query = "          UPDATE radacct           SET
acctstoptime       =  '%S',             acctsessiontime    =  unix_timestamp('%S')
-                                  unix_timestamp(acctstarttime),
acctterminatecause =  '%{Acct-Terminate-Cause}',            acctstopdelay      =
%{%{Acct-Delay-Time}:-0}           WHERE acctstoptime IS NULL           AND
nasipaddress      =  '%{NAS-IP-Address}'           AND acctstarttime     <= '%S'"
    accounting_update_query = "           UPDATE radacct           SET
framedipaddress = '%{Framed-IP-Address}',             acctsessiontime     =
'%{Acct-Session-Time}',            acctinputoctets     =
'%{%{Acct-Input-Gigawords}:-0}'  << 32 |
'%{%{Acct-Input-Octets}:-0}',             acctoutputoctets    =
'%{%{Acct-Output-Gigawords}:-0}' << 32 |
'%{%{Acct-Output-Octets}:-0}'           WHERE acctsessionid = '%{Acct-Session-Id}'
AND username        = '%{SQL-User-Name}'           AND nasipaddress    =
'%{NAS-IP-Address}'"
    accounting_update_query_alt = "           INSERT INTO radacct
(acctsessionid,    acctuniqueid,      username,             realm,
nasipaddress,      nasportid,              nasporttype,      acctstarttime,
acctsessiontime,             acctauthentic,    connectinfo_start, acctinputoctets,
acctoutputoctets, calledstationid,   callingstationid,            servicetype,
framedprotocol,    framedipaddress,             acctstartdelay,
xascendsessionsvrkey)           VALUES             ('%{Acct-Session-Id}',
'%{Acct-Unique-Session-Id}',              '%{SQL-User-Name}',
'%{Realm}', '%{NAS-IP-Address}', '%{NAS-Port}',              '%{NAS-Port-Type}',
DATE_SUB('%S',                     INTERVAL (%{%{Acct-Session-Time}:-0} +
%{%{Acct-Delay-Time}:-0}) SECOND),                     '%{Acct-Session-Time}',
'%{Acct-Authentic}', '',              '%{%{Acct-Input-Gigawords}:-0}' << 32 |
'%{%{Acct-Input-Octets}:-0}',              '%{%{Acct-Output-Gigawords}:-0}' << 32 |
'%{%{Acct-Output-Octets}:-0}',              '%{Called-Station-Id}',
'%{Calling-Station-Id}',             '%{Service-Type}', '%{Framed-Protocol}',
'%{Framed-IP-Address}',             '0', '%{X-Ascend-Session-Svr-Key}')"
    accounting_start_query = "          INSERT INTO radacct
(acctsessionid,    acctuniqueid,     username,             realm,
nasipaddress,     nasportid,              nasporttype,      acctstarttime,
acctstoptime,             acctsessiontime,  acctauthentic,    connectinfo_start,
connectinfo_stop, acctinputoctets,  acctoutputoctets,            calledstationid,
callingstationid, acctterminatecause,            servicetype,       framedprotocol,
framedipaddress,             acctstartdelay,   acctstopdelay,
xascendsessionsvrkey)           VALUES             ('%{Acct-Session-Id}',
'%{Acct-Unique-Session-Id}',              '%{SQL-User-Name}',
'%{Realm}', '%{NAS-IP-Address}', '%{NAS-Port}',             '%{NAS-Port-Type}',
'%S', NULL,              '0', '%{Acct-Authentic}', '%{Connect-Info}',
'', '0', '0',              '%{Called-Station-Id}', '%{Calling-Station-Id}', '',
'%{Service-Type}', '%{Framed-Protocol}', '%{Framed-IP-Address}',
'%{%{Acct-Delay-Time}:-0}', '0', '%{X-Ascend-Session-Svr-Key}')"
    accounting_start_query_alt = "          UPDATE radacct SET
acctstarttime     = '%S',              acctstartdelay    =
'%{%{Acct-Delay-Time}:-0}',              connectinfo_start = '%{Connect-Info}'
WHERE acctsessionid  = '%{Acct-Session-Id}'           AND username         =
'%{SQL-User-Name}'            AND nasipaddress     = '%{NAS-IP-Address}'"
    accounting_stop_query = "           UPDATE radacct SET              acctstoptime
= '%S',               acctsessiontime    = '%{Acct-Session-Time}',
acctinputoctets    = '%{%{Acct-Input-Gigawords}:-0}' << 32 |
'%{%{Acct-Input-Octets}:-0}',              acctoutputoctets   =
'%{%{Acct-Output-Gigawords}:-0}' << 32 |
'%{%{Acct-Output-Octets}:-0}',              acctterminatecause =
'%{Acct-Terminate-Cause}',              acctstopdelay      =
'%{%{Acct-Delay-Time}:-0}',              connectinfo_stop   = '%{Connect-Info}'
WHERE acctsessionid   = '%{Acct-Session-Id}'           AND username          =
'%{SQL-User-Name}'            AND nasipaddress      = '%{NAS-IP-Address}'"
    accounting_stop_query_alt = "          INSERT INTO radacct
(acctsessionid, acctuniqueid, username,              realm, nasipaddress, nasportid,
nasporttype, acctstarttime, acctstoptime,            acctsessiontime,
acctauthentic, connectinfo_start,              connectinfo_stop, acctinputoctets,
acctoutputoctets,              calledstationid, callingstationid, acctterminatecause,
servicetype, framedprotocol, framedipaddress,             acctstartdelay,
acctstopdelay)           VALUES            ('%{Acct-Session-Id}',
'%{Acct-Unique-Session-Id}',              '%{SQL-User-Name}',
'%{Realm}', '%{NAS-IP-Address}', '%{NAS-Port}',             '%{NAS-Port-Type}',
DATE_SUB('%S',                  INTERVAL (%{%{Acct-Session-Time}:-0} +
%{%{Acct-Delay-Time}:-0}) SECOND),              '%S', '%{Acct-Session-Time}',
'%{Acct-Authentic}', '',              '%{Connect-Info}',
'%{%{Acct-Input-Gigawords}:-0}' << 32 |              '%{%{Acct-Input-Octets}:-0}',
'%{%{Acct-Output-Gigawords}:-0}' << 32 |             '%{%{Acct-Output-Octets}:-0}',
'%{Called-Station-Id}', '%{Calling-Station-Id}',
'%{Acct-Terminate-Cause}',              '%{Service-Type}', '%{Framed-Protocol}',
'%{Framed-IP-Address}',              '0', '%{%{Acct-Delay-Time}:-0}')"
    group_membership_query = "SELECT groupname          FROM radusergroup
WHERE username = '%{SQL-User-Name}'           ORDER BY priority"
    connect_failure_retry_delay = 60
    simul_count_query = ""
    simul_verify_query = "SELECT radacctid, acctsessionid, username,
nasipaddress, nasportid, framedipaddress,
callingstationid, framedprotocol                              FROM radacct
```

```
WHERE username = '%{SQL-User-Name}'                              AND acctstoptime
IS NULL"
    postauth_query = "INSERT INTO radpostauth                           (username,
pass, reply, authdate)                            VALUES
(                            '%{User-Name}',
'%{%{User-Password}:-%{Chap-Password}}',
'%{reply:Packet-Type}', '%S')"
    safe-characters =
"@abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.-_: /"
  }
rlm_sql (sql): Driver rlm_sql_MySQL (module rlm_sql_MySQL) loaded and linked
rlm_sql (sql): Attempting to connect to radius@localhost:/radius
rlm_sql (sql): starting 0
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #0
rlm_sql_MySQL: Starting connect to MySQL server for #0
rlm_sql (sql): Connected new DB handle, #0
rlm_sql (sql): starting 1
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #1
rlm_sql_MySQL: Starting connect to MySQL server for #1
rlm_sql (sql): Connected new DB handle, #1
rlm_sql (sql): starting 2
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #2
rlm_sql_MySQL: Starting connect to MySQL server for #2
rlm_sql (sql): Connected new DB handle, #2
rlm_sql (sql): starting 3
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #3
rlm_sql_MySQL: Starting connect to MySQL server for #3
rlm_sql (sql): Connected new DB handle, #3
rlm_sql (sql): starting 4
rlm_sql (sql): Attempting to connect rlm_sql_MySQL #4
rlm_sql_MySQL: Starting connect to MySQL server for #4
rlm_sql (sql): Connected new DB handle, #4
 Module: Checking preacct {...} for more modules to load
 Module: Linked to module rlm_acct_unique
 Module: Instantiating acct_unique
  acct_unique {
    key = "User-Name, Acct-Session-Id, NAS-IP-Address, Client-IP-Address, NAS-Port"
  }
 Module: Checking accounting {...} for more modules to load
 Module: Linked to module rlm_detail
 Module: Instantiating detail
  detail {
    detailfile =
"/usr/local/var/log/radius/radacct/%{Client-IP-Address}/detail-%Y%m%d"
```

```
    header = "%t"
    detailperm = 384
    dirperm = 493
    locking = no
    log_packet_header = no
  }
 Module: Instantiating attr_filter.accounting_response
  attr_filter attr_filter.accounting_response {
    attrsfile = "/usr/local/etc/raddb/attrs.accounting_response"
    key = "%{User-Name}"
  }
 Module: Checking session {...} for more modules to load
 Module: Checking post-proxy {...} for more modules to load
 Module: Checking post-auth {...} for more modules to load
 }
}
radiusd: #### Opening IP addresses and Ports ####
listen {
    type = "auth"
    ipaddr = *
    port = 0
}
listen {
    type = "acct"
    ipaddr = *
    port = 0
}
Listening on authentication address * port 1812
Listening on accounting address * port 1813
Listening on proxy address * port 1814
Ready to process requests.
rad_recv: Access-Request packet from host 127.0.0.1 port 1029, id=26, length=57
    User-Name = "fredf"
    User-Password = "nikos"
    NAS-IP-Address = 127.0.0.2
    NAS-Port = 1812
+- entering group authorize
++[preprocess] returns ok
++[chap] returns noop
++[mschap] returns noop
    rlm_realm: No '@' in User-Name = "fredf", looking up realm NULL
    rlm_realm: No such realm "NULL"
++[suffix] returns noop
  rlm_eap: No EAP-Message, not doing EAP
```

```
++[eap] returns noop
++[unix] returns notfound
++[files] returns noop
    expand: %{User-Name} -> fredf
rlm_sql (sql): sql_set_user escaped user --> 'fredf'
rlm_sql (sql): Reserving sql socket id: 4
    expand: SELECT id, username, attribute, value, op           FROM radcheck
WHERE username = '%{SQL-User-Name}'           ORDER BY id -> SELECT id, username,
attribute, value, op           FROM radcheck           WHERE username = 'fredf'
ORDER BY id
rlm_sql (sql): User found in radcheck table
    expand: SELECT id, username, attribute, value, op           FROM radreply
WHERE username = '%{SQL-User-Name}'           ORDER BY id -> SELECT id, username,
attribute, value, op           FROM radreply           WHERE username = 'fredf'
ORDER BY id
    expand: SELECT groupname           FROM radusergroup           WHERE username =
'%{SQL-User-Name}'           ORDER BY priority -> SELECT groupname           FROM
radusergroup           WHERE username = 'fredf'           ORDER BY priority
rlm_sql (sql): Released sql socket id: 4
++[sql] returns ok
++[expiration] returns noop
++[logintime] returns noop
++[pap] returns updated
  rad_check_password:  Found Auth-Type
auth: type "PAP"
+- entering group PAP
rlm_pap: login attempt with password "nikos"
rlm_pap: Using clear text password "fredf"
rlm_pap: Passwords don't match
++[pap] returns reject
auth: Failed to validate the user.
  Found Post-Auth-Type Reject
+- entering group REJECT
    expand: %{User-Name} -> fredf
 attr_filter: Matched entry DEFAULT at line 11
++[attr_filter.access_reject] returns updated
Sending Access-Reject of id 26 to 127.0.0.1 port 1029
    Reply-Message := "Hello fredf!"
Finished request 0.
Going to the next request
Waking up in 4.9 seconds.
Cleaning up request 0 ID 26 with timestamp +5
Ready to process requests.
```

Appendix VI. RADIUS Packet Type Code

#	Message	Reference
----	------------------------	---------
1	Access-Request	[RFC2865]
2	Access-Accept	[RFC2865]
3	Access-Reject	[RFC2865]
4	Accounting-Request	[RFC2865]
5	Accounting-Response	[RFC2865]
6	Accounting-Status (now Interim Accounting)	[RFC2882]
7	Password-Request	[RFC2882]
8	Password-Ack	[RFC2882]
9	Password-Reject	[RFC2882]
10	Accounting-Message	[RFC2882]
11	Access-Challenge	[RFC2865]
12	Status-Server (experimental)	[RFC2865]
13	Status-Client (experimental)	[RFC2865]
21	Resource-Free-Request	[RFC2882]
22	Resource-Free-Response	[RFC2882]
23	Resource-Query-Request	[RFC2882]
24	Resource-Query-Response	[RFC2882]
25	Alternate-Resource-Reclaim-Request	[RFC2882]
26	NAS-Reboot-Request	[RFC2882]
27	NAS-Reboot-Response	[RFC2882]
28	Reserved	
29	Next-Passcode	[RFC2882]
30	New-Pin	[RFC2882]

31	Terminate-Session	[RFC2882]
32	Password-Expired	[RFC2882]
33	Event-Request	[RFC2882]
34	Event-Response	[RFC2882]
40	Disconnect-Request	[RFC3575]
41	Disconnect-ACK	[RFC3575]
42	Disconnect-NAK	[RFC3575]
43	CoA-Request	[RFC3575]
44	CoA-ACK	[RFC3575]
45	CoA-NAK	[RFC3575]
50	IP-Address-Allocate	[RFC2882]
51	IP-Address-Release	[RFC2882]
250-253	Experimental Use	
254	Reserved	
255	Reserved	[RFC2865]

Appendix VII. RADIUS Attribute type

(Defined in RFC 2865)

VALUE	Description	Reference
------	-----------	--------
1	User-Name	
2	User-Password	
3	CHAP-Password	
4	NAS-IP-Address	
5	NAS-Port	
6	Service-Type	
7	Framed-Protocol	
8	Framed-IP-Address	
9	Framed-IP-Netmask	
10	Framed-Routing	
11	Filter-Id	
12	Framed-MTU	
13	Framed-Compression	
14	Login-IP-Host	
15	Login-Service	
16	Login-TCP-Port	
17	(unassigned)	
18	Reply-Message	
19	Callback-Number	
20	Callback-Id	
21	(unassigned)	
22	Framed-Route	
23	Framed-IPX-Network	
24	State	

25	Class
26	Vendor-Specific
27	Session-Timeout
28	Idle-Timeout
29	Termination-Action
30	Called-Station-Id
31	Calling-Station-Id
32	NAS-Identifier
33	Proxy-State
34	Login-LAT-Service
35	Login-LAT-Node
36	Login-LAT-Group
37	Framed-AppleTalk-Link
38	Framed-AppleTalk-Network
39	Framed-AppleTalk-Zone
40	Acct-Status-Type [RFC2866]
41	Acct-Delay-Time [RFC2866]
42	Acct-Input-Octets [RFC2866]
43	Acct-Output-Octets [RFC2866]
44	Acct-Session-Id [RFC2866]
45	Acct-Authentic [RFC2866]
46	Acct-Session-Time [RFC2866]
47	Acct-Input-Packets [RFC2866]
48	Acct-Output-Packets [RFC2866]
49	Acct-Terminate-Cause [RFC2866]
50	Acct-Multi-Session-Id [RFC2866]
51	Acct-Link-Count [RFC2866]
52	Acct-Input-Gigawords [RFC2869]
53	Acct-Output-Gigawords [RFC2869]
54	(unassigned)

55	Event-Timestamp [RFC2869]
56-59	(unassigned)
60	CHAP-Challenge
61	NAS-Port-Type
62	Port-Limit
63	Login-LAT-Port
64	Tunnel-Type [RFC2868]
65	Tunnel-Medium-Type [RFC2868]
66	Tunnel-Client-Endpoint [RFC2868]
67	Tunnel-Server-Endpoint [RFC2868]
68	Acct-Tunnel-Connection [RFC2867]
69	Tunnel-Password [RFC2868]
70	ARAP-Password [RFC2869]
71	ARAP-Features [RFC2869]
72	ARAP-Zone-Access [RFC2869]
73	ARAP-Security [RFC2869]
74	ARAP-Security-Data [RFC2869]
75	Password-Retry [RFC2869]
76	Prompt [RFC2869]
77	Connect-Info [RFC2869]
78	Configuration-Token [RFC2869]
79	EAP-Message [RFC2869]
80	Message-Authenticator [RFC2869]
81	Tunnel-Private-Group-ID [RFC2868]
82	Tunnel-Assignment-ID [RFC2868]
83	Tunnel-Preference [RFC2868]
84	ARAP-Challenge-Response [RFC2869]
85	Acct-Interim-Interval [RFC2869]
86	Acct-Tunnel-Packets-Lost [RFC2867]
87	NAS-Port-Id [RFC2869]

88	Framed-Pool [RFC2869]	
89	CUI [RFC-ietf-radext-chargeable-user-id-06.txt]	
90	Tunnel-Client-Auth-ID [RFC2868]	
91	Tunnel-Server-Auth-ID [RFC2868]	
92-93	(Unassigned)	
94	Originating-Line-Info	[Trifunovic]
95	NAS-IPv6-Address [RFC3162]	
96	Framed-Interface-Id [RFC3162]	
97	Framed-IPv6-Prefix [RFC3162]	
98	Login-IPv6-Host [RFC3162]	
99	Framed-IPv6-Route [RFC3162]	
100	Framed-IPv6-Pool [RFC3162]	
101	Error-Cause Attribute	[RFC3576]
102	EAP-Key-Name	[RFC4072]
103	Digest-Response	[RFC-ietf-radext-digest-auth-09.txt]
104	Digest-Realm [RFC-ietf-radext-digest-auth-09.txt]	
105	Digest-Nonce	[RFC-ietf-radext-digest-auth-09.txt]
106	Digest-Nextnonce	[RFC-ietf-radext-digest-auth-09.txt]
107	Digest-Response-Auth	[RFC-ietf-radext-digest-auth-09.txt]
108	Digest-Method	[RFC-ietf-radext-digest-auth-09.txt]
109	Digest-URI	[RFC-ietf-radext-digest-auth-09.txt]
110	Digest-Qop	[RFC-ietf-radext-digest-auth-09.txt]
111	Digest-Algorithm	[RFC-ietf-radext-digest-auth-09.txt]
112	Digest-Entity-Body-Hash	[RFC-ietf-radext-digest-auth-09.txt]
113	Digest-CNonce	[RFC-ietf-radext-digest-auth-09.txt]
114	Digest-Nonce-Count	[RFC-ietf-radext-digest-auth-09.txt]
115	Digest-Username	[RFC-ietf-radext-digest-auth-09.txt]
116	Digest-Opaque	[RFC-ietf-radext-digest-auth-09.txt]
117	Digest-Auth-Param	[RFC-ietf-radext-digest-auth-09.txt]
118	Digest-AKA-Auts	[RFC-ietf-radext-digest-auth-09.txt]

119	Digest-Domain	[RFC-ietf-radext-digest-auth-09.txt]
120	Digest-Stale	[RFC-ietf-radext-digest-auth-09.txt]
121	Digest-HA1	[RFC-ietf-radext-digest-auth-09.txt]
122	SIP-AOR	[RFC-ietf-radext-digest-auth-09.txt]
123-191	(unassigned)	

192-223 Experimental Use [RFC2058]

224-240 Implementation Specific [RFC2058]

241-255 Reserved [RFC2058]

RADIUS Attribute Values

Defined in RFC 2865 unless otherwise indicated.

Values for RADIUS Attribute 6, Service-Type:

1	Login	
2	Framed	
3	Callback Login	
4	Callback Framed	
5	Outbound	
6	Administrative	
7	NAS Prompt	
8	Authenticate Only	
9	Callback NAS Prompt	
10	Call Check	
11	Callback Administrative	
12	Voice	[Chiba]
13	Fax	[Chiba]
14	Modem Relay	[Chiba]
15	IAPP-Register	[IEEE 802.11f][Kerry]
16	IAPP-AP-Check	[IEEE 802.11f][Kerry]
17	Authorize Only	[RFC3576]

Values for RADIUS Attribute 7, Framed-Protocol:

1 PPP

2 SLIP

3 AppleTalk Remote Access Protocol (ARAP)

4 Gandalf proprietary SingleLink/MultiLink protocol

5 Xylogics proprietary IPX/SLIP

6 X.75 Synchronous

7 GPRS PDP Context [Moore]

Values for RADIUS Attribute 10, Framed-Routing:

0 None

1 Send routing packets

2 Listen for routing packets

3 Send and Listen

Values for RADIUS Attribute 13, Framed-Compression:

0 None

1 VJ TCP/IP header compression

2 IPX header compression

3 Stac-LZS compression

Values for RADIUS Attribute 15, Login-Service:

0 Telnet

1 Rlogin

2 TCP Clear

3 PortMaster (proprietary)

4 LAT

5 X25-PAD

6 X25-T3POS

7 (unassigned)

8 TCP Clear Quiet (suppresses any NAS-generated connect string)

Values for RADIUS Attribute 29, Termination-Action:

0 Default

1 RADIUS-Request

Values for RADIUS Attribute 40, Acct-Status-Type [RFC 2866]:

1 Start [RFC 2866]

2 Stop [RFC 2866]

3 Interim-Update [RFC 2866]

4-6 (unassigned)

7 Accounting-On [RFC 2866]

8 Accounting-Off [RFC 2866]

9 Tunnel-Start [RFC 2867]

10 Tunnel-Stop [RFC 2867]

11 Tunnel-Reject [RFC 2867]

12 Tunnel-Link-Start [RFC 2867]

13 Tunnel-Link-Stop [RFC 2867]

14 Tunnel-Link-Reject [RFC 2867]

15 Failed [RFC 2866]

Values for RADIUS Attribute 45, Acct-Authentic [RFC 2866]:

1 RADIUS

2 Local

3 Remote

4 Diameter   [Calhoun]

Values for RADIUS Attribute 49, Acct-Terminate-Cause [RFC 2866]:

1 User Request

2	Lost Carrier	
3	Lost Service	
4	Idle Timeout	
5	Session Timeout	
6	Admin Reset	
7	Admin Reboot	
8	Port Error	
9	NAS Error	
10	NAS Request	
11	NAS Reboot	
12	Port Unneeded	
13	Port Preempted	
14	Port Suspended	
15	Service Unavailable	
16	Callback	
17	User Error	
18	Host Request	
19	Supplicant Restart	[RFC3580]
20	Reauthentication Failure	[RFC3580]
21	Port Reinitialized	[RFC3580]
22	Port Administratively Disabled	[RFC3580]

Values for RADIUS Attribute 61, NAS-Port-Type [RFC 2865]:

0	Async
1	Sync
2	ISDN Sync
3	ISDN Async V.120
4	ISDN Async V.110
5	Virtual
6	PIAFS

7 HDLC Clear Channel

8 X.25

9 X.75

10 G.3 Fax

11 SDSL - Symmetric DSL

12 ADSL-CAP - Asymmetric DSL, Carrierless Amplitude Phase Modulation

13 ADSL-DMT - Asymmetric DSL, Discrete Multi-Tone

14 IDSL - ISDN Digital Subscriber Line

15 Ethernet

16 xDSL - Digital Subscriber Line of unknown type

17 Cable

18 Wireless - Other

19 Wireless - IEEE 802.11

20 Token-Ring [RFC3580]

21 FDDI [RFC3580]

22 Wireless - CDMA2000 [McCann]

23 Wireless - UMTS [McCann]

24 Wireless - 1X-EV [McCann]

25 IAPP [IEEE 802.11f][Kerry]

26 FTTP - Fiber to the Premises [Nyce]

27-29 Unassigned

30 PPPoA - PPP over ATM [RFC-zorn-radius-port-type-04.txt]

31 PPPoEoA - PPP over Ethernet over ATM [RFC-zorn-radius-port-type-04.txt]

32 PPPoEoE - PPP over Ethernet over Ethernet [RFC-zorn-radius-port-type-04.txt]

33 PPPoEoVLAN - PPP over Ethernet over VLAN [RFC-zorn-radius-port-type-04.txt]

34 PPPoEoQinQ - PPP over Ethernet over IEEE 802.1QinQ [RFC-zorn-radius-port-type-04.txt]

Values for RADIUS Attribute 64, Tunnel-Type [RFC 2868]:

1 Point-to-Point Tunneling Protocol (PPTP)

2 Layer Two Forwarding (L2F)

3 Layer Two Tunneling Protocol (L2TP)

4 Ascend Tunnel Management Protocol (ATMP)

5 Virtual Tunneling Protocol (VTP)

6 IP Authentication Header in the Tunnel-mode (AH)

7 IP-in-IP Encapsulation (IP-IP)

8 Minimal IP-in-IP Encapsulation (MIN-IP-IP)

9 IP Encapsulating Security Payload in the Tunnel-mode (ESP)

10 Generic Route Encapsulation (GRE)

11 Bay Dial Virtual Services (DVS)

12 IP-in-IP Tunneling

13 Virtual LANs (VLAN) [RFC3580]

Values for RADIUS Attribute 65, Tunnel-Medium-Type [RFC 2868]:

1 IPv4 (IP version 4)

2 IPv6 (IP version 6)

3 NSAP

4 HDLC (8-bit multidrop)

5 BBN 1822

6 802 (includes all 802 media plus Ethernet "canonical format")

7 E.163 (POTS)

8 E.164 (SMDS, Frame Relay, ATM)

9 F.69 (Telex)

10 X.121 (X.25, Frame Relay)

11 IPX

12 Appletalk

13 Decnet IV

14 Banyan Vines

15 E.164 with NSAP format subaddress

Values for RADIUS Attribute 72, ARAP-Zone-Access [RFC 2869]:

1 Only allow access to default zone

2 Use zone filter inclusively

3 (not used)

4 Use zone filter exclusively

Values for RADIUS Attribute 76, Prompt [RFC 2869]:

0 No Echo

1 Echo

Values for RADIUS Attribute 101, Error-Cause Attribute [RFC3576]:

201	Residual Session Context Removed
202	Invalid EAP Packet (Ignored)
401	Unsupported Attribute
402	Missing Attribute
403	NAS Identification Mismatch
404	Invalid Request
405	Unsupported Service
406	Unsupported Extension
501	Administratively Prohibited
502	Request Not Routable (Proxy)
503	Session Context Not Found
504	Session Context Not Removable
505	Other Proxy Processing Error
506	Resources Unavailable

507 Request Initiated

COS/CCS 2008-19