Radius Macadressenkontrolle für WLAN über LDAPauth UCS (Univention) mit Fortinet Accesspoints

Du möchtest dich gerne für unsere Hilfe erkenntlich zeigen . Gerne. Wir bedanken uns bei dir für deine Spende!

Spenden

Hauseigenes Apt-Repo: https://apt.iteas.at

GITLAB Enterprise:

In diesem HowTo beschreibe ich wie man zusätzlich zur WLAN WPA2/3 Enterprise Auth. mit UCS (Univention) LDAP auch eine Macadressenkontrolle mit Radius umsetzen kann. Als Accesspoints verwenden wir hier FortiAP's. Das ganze hat den Vorteil das man mit Radius noch eine weiter Sicherheitsschicht einführt. Somit muss der Client am LDAP mit einem Computerkonto eingetragen sein. Ist er das nicht, ist trotz richtigen Authentifizierungsdaten kein Login am WLAN möglich.

Folgende OS Versionen wurden eingesetzt:
4.4-8 errata1019
5.0-2 errata366 (Upgradeanleitung beachten)
FortiOS v7.0.1
ForitAP v7.0.1

Konfiguration

Voraussetzung ist hier das man sich bereits erfolgreich am WLAN mit WPA2/3 Enterprise anmelden kann, wenn man in einer ausgewählten LDAP-Gruppe des UCS-System Mitglied ist.

WLAN Fortinet

Um nun die MAC-Kontrolle für eine SSID zu aktivieren, geht man folgender Maßen for:

```
config wireless-controller vap
    edit "mywlanssid"
        set ssid "mywlanssid"
     +  set mac-username-delimiter colon
     +  set mac-password-delimiter colon
        set security wpa2-only-enterprise
```

```
        set pmf enable
     +  set radius-mac-auth enable
     +  set radius-mac-auth-server "UCS-Radius"
        set auth usergroup
        set local-bridging enable
        set usergroup "wifi-wlan"
        set schedule "always"
        set vlanid 44
    next
end
```

Essentiell sind die Zeilen mit dem „+“. Der Name des auth-server kann natürlich abweichen.

Konfiguration Univention UCS

Hierfür sind einige Dinge zu beachten. Zum einen muß die Funktion für die Macadressenkontrolle aktiviert werden:

```
usr set radius/mac/whitelisting=true
```

Weiters muss ein Filter im LDAP Modul von Radius verändert werden. Vorher legen wir noch kurz ein Backup der Dateien an:

```
cp  /etc/univention/templates/files/etc/freeradius/3.0/mods-available/ldap
/etc/univention/templates/files/etc/freeradius/3.0/mods-
available/ldap_backup_orig
cp  /etc/freeradius/3.0/mods-enabled/ldap /etc/freeradius/3.0/mods-
enabled/ldap_backup_orig
```

Nun die Änderungen durchführen:

```
nano /etc/univention/templates/files/etc/freeradius/3.0/mods-available/ldap

@!@
auth_type = configRegistry.get('freeradius/conf/auth-type/mschap', 'FALSE')
if auth_type and 'TRUE' == auth_type.upper() or 'YES' == auth_type.upper():
#else:
#       filter = 'Stripped-User-Name'
#print '\t\tfilter = "(uid=%%{%s:-%%{User-Name}})"' % filter
@!@
filter = "(|(uid=%{mschap:User-Name:-%{User-
Name}})(macAddress=%{mschap:User-Name:-%{User-Name}}))"
```

Nun noch die Änderungen in die Konfiguration übernehmen und den Radius neu starten. Danach den Radius neu starten:

```
ucr commit /etc/freeradius/3.0/mods-available/ldap && service freeradius
restart
```

Die Änderungen müssen natürlich auf allen Radiusservern im Netzwerk durchgeführt werden.

Zu guter letzt ist es noch wichtig für die Konten den Gerätetyp „Linux“ zu verwenden. Die WLANclients benötigen einen vollwertigen Computeraccount. Unter „Allgemein“ muss **Rechnername, MAC Adresse und IP** angegeben werden. Unter „Radius“ ein Hakerl bei **Netzwerkzugriff erlaubt** setzen.

Danach muss man unter „Erweiterte Einstellungen“ → Konto, das Gerätepasswort eintragen. Dies ist die MAC-Adresse des Gerätes in Großbuchstaben.
24-EF-BA-96-D2-03 → Falsch
24:ef:ba:96:d2:03 → Falsch
24:EF:BA:96:D2:03 → Richtig

Und schon ist die Macadressenkontrolle aktiv.

Upgrade UCS5

Das File `/etc/univention/templates/files/etc/freeradius/3.0/mods-available/ldap` ist der Zeit noch nicht mit dem Upgradeprozess kompatibel. — *2022/07/25 20:58* Daher muss das aktuelle template von einer frischen Raidusinstallation von UCS kopiert werden. Das Files sind hier ganz am Ende der Seite angehängt. Einmal „ldap“ → fertigs file, einfach einspielen und commiten. ldap_orig_UCS5 → unverändertes File von UCS5.

Clientfalle

Ändert man das Device/Gerätekontenpasswort in UCS auf die Macadrresse, muss dieses natürlich auch auf dem Client in der sssd.conf nachgetragen werden. Auch ein erneuter Export der Keytab ist erforderlich.

Radius Debug

Wichtig ist hier zu erwähnen das dies je nach Router/WLAN das man verwendet etwas anders sein kann. Um zu erfahren welches Passwort der Client nun wirklich mitsendet. Stoppt man Radius und startet ihn im Debugmode neu:

```
service freeradius stop
freeradius -X
```

Nun lässt man einen Client per WLAN verbinden. Sämtliche Anfragen und Logs sieht man nun live in dieser Ausgabe, auch welches Passwort vom Client mit gesendet wird.

UCS 4.4 kommt mit einer verbesserten Fehlersuche. Mit dem Kommandozeilentool `univention-radius-check-access` können Sie aktuelle Zugangsregeln für einen bestimmten Benutzer und/oder eine MAC-Adresse überprüfen. Sie rufen den Befehl als Benutzer root auf dem UCS-Server (in einem Terminalfenster oder auf der Konsole) auf. Die RADIUS-App protokolliert die Ereignisse und schreibt sie in die Logdatei `/var/log/univention/radius_ntlm_auth.log`. Wie ausführlich die Meldungen sind, legen Sie über die Univention-Configuration-Registry-Variable

`freeradius/auth/helper/ntlm/debug` fest. Der FreeRADIUS-Server legt ebenfalls seine eigene Logdatei unter `/var/log/freeradius/radius.log` an.

Files für UCS5 Upgrade

ldap

```
@%@UCRWARNING=# @%@
# -*- text -*-
#
#  $Id: 4b7e4585c029b8617aa7b9169a42bf50a5ec4938 $

#
#  Lightweight Directory Access Protocol (LDAP)
#
ldap {
        #  Note that this needs to match the name(s) in the LDAP server
        #  certificate, if you're using ldaps.  See OpenLDAP
documentation
        #  for the behavioral semantics of specifying more than one
host.
        #
        #  Depending on the libldap in use, server may be an LDAP URI.
        #  In the case of OpenLDAP this allows additional the following
        #  additional schemes:
        #  - ldaps:// (LDAP over SSL)
        #  - ldapi:// (LDAP over Unix socket)
        #  - ldapc:// (Connectionless LDAP)
        server = "@%@ldap/server/name@%@"
#       server = 'ldap.rrdns.example.org'
#       server = 'ldap.rrdns.example.org'

        #  Port to connect on, defaults to 389, will be ignored for
LDAP URIs.
#       port = 389
@!@
print('\tport = "%s"' % (configRegistry.get('ldap/server/port',
'7389'), ))
@!@

        #  Administrator account for searching and possibly modifying.
        #  If using SASL + KRB5 these should be commented out.
#       identity = 'cn=admin,dc=example,dc=org'
        identity = "@%@ldap/hostdn@%@"
#       password = mypass
@!@
with open('/etc/machine.secret', 'r') as credfile:
        print("\tpassword = \"%s\"" % credfile.readline().rstrip("\n"))
@!@
```

```
        #  Unless overridden in another section, the dn from which all
        #  searches will start from.
#       base_dn = 'dc=example,dc=org'
        base_dn = "@%@ldap/base@%@"

        #
        #  SASL parameters to use for admin binds
        #
        #  When we're prompted by the SASL library, these control
        #  the responses given, as well as the identity and password
        #  directives above.
        #
        #  If any directive is commented out, a NULL response will be
        #  provided to cyrus-sasl.
        #
        #  Unfortunately the only way to control Keberos here is
through
        #  environmental variables, as cyrus-sasl provides no API to
        #  set the krb5 config directly.
        #
        #  Full documentation for MIT krb5 can be found here:
        #
        #
http://web.mit.edu/kerberos/krb5-devel/doc/admin/env_variables.html
        #
        #  At a minimum you probably want to set KRB5_CLIENT_KTNAME.
        #
        sasl {
                # SASL mechanism
#               mech = 'PLAIN'

                # SASL authorisation identity to proxy.
#               proxy = 'autz_id'

                # SASL realm. Used for kerberos.
#               realm = 'example.org'
        }

        #
        #  Generic valuepair attribute
        #

        #  If set, this will attribute will be retrieved in addition to
any
        #  mapped attributes.
        #
        #  Values should be in the format:
        #       <radius attr> <op> <value>
        #
        #  Where:
```

```
        #       <radius attr>:  Is the attribute you wish to create
        #                       with any valid list and request
qualifiers.
        #       <op>:           Is any assignment operator (=, :=, +=,
-=).
        #       <value>:        Is the value to parse into the new
valuepair.
        #                       If the value is wrapped in double
quotes it
        #                       will be xlat expanded.
#       valuepair_attribute = 'radiusAttribute'

        #
        #  Mapping of LDAP directory attributes to RADIUS dictionary
attributes.
        #

        #  WARNING: Although this format is almost identical to the
unlang
        #  update section format, it does *NOT* mean that you can use
other
        #  unlang constructs in module configuration files.
        #
        #  Configuration items are in the format:
        #       <radius attr> <op> <ldap attr>
        #
        #  Where:
        #       <radius attr>:  Is the destination RADIUS attribute
        #                       with any valid list and request
qualifiers.
        #       <op>:           Is any assignment attribute (=, :=, +=,
-=).
        #       <ldap attr>:    Is the attribute associated with user
or
        #                       profile objects in the LDAP directory.
        #                       If the attribute name is wrapped in
double
        #                       quotes it will be xlat expanded.
        #
        #  Request and list qualifiers may also be placed after the
'update'
        #  section name to set defaults destination requests/lists
        #  for unqualified RADIUS attributes.
        #
        #  Note: LDAP attribute names should be single quoted unless
you want
        #  the name value to be derived from an xlat expansion, or an
        #  attribute ref.
        update {
                control:Password-With-Header    += 'userPassword'
#               control:NT-Password             := 'ntPassword'
```

```
#               reply:Reply-Message             := 'radiusReplyMessage'
#               reply:Tunnel-Type               := 'radiusTunnelType'
#               reply:Tunnel-Medium-Type        :=
'radiusTunnelMediumType'
#               reply:Tunnel-Private-Group-ID   :=
'radiusTunnelPrivategroupId'

                #  Where only a list is specified as the RADIUS
attribute,
                #  the value of the LDAP attribute is parsed as a
valuepair
                #  in the same format as the 'valuepair_attribute'
(above).
                control:                        +=
'radiusControlAttribute'
                request:                        +=
'radiusRequestAttribute'
                reply:                          +=
'radiusReplyAttribute'
@!@
auth_type = configRegistry.get('freeradius/conf/auth-type/mschap',
'FALSE')

if auth_type and 'TRUE' == auth_type.upper() or 'YES' ==
auth_type.upper():
        print('control:LM-Password\t\t:=\tsambaLMPassword')
        print('control:NT-Password\t\t:=\tsambaNTPassword')
else:
        print('control:LM-Password\t\t:=\tlmPassword')
        print('control:NT-Password\t\t:=\tntPassword')
        print('control:LM-Password\t\t:=\tsambaLmPassword')
        print('control:NT-Password\t\t:=\tsambaNtPassword')
@!@
        }

        #  Set to yes if you have eDirectory and want to use the
universal
        #  password mechanism.
#       edir = no

        #  Set to yes if you want to bind as the user after retrieving
the
        #  Cleartext-Password. This will consume the login grace, and
        #  verify user authorization.
#       edir_autz = no

        #  Note: set_auth_type was removed in v3.x.x
        #  Equivalent functionality can be achieved by adding the
following
        #  stanza to the authorize {} section of your virtual server.
        #
```

```
        #    ldap
        #    if ((ok || updated) && User-Password) {
        #        update {
        #            control:Auth-Type := ldap
        #        }
        #    }

        #
        #  User object identification.
        #
        user {
                #  Where to start searching in the tree for users
                base_dn = "${..base_dn}"

                #  Filter for user objects, should be specific enough
                #  to identify a single user object.
                #
                #  For Active Directory, you should use
                #  "samaccountname=" instead of "uid="
                #
#               filter = "(uid=%{%{Stripped-User-Name}:-%{User-Name}})"
@!@
auth_type = configRegistry.get('freeradius/conf/auth-type/mschap',
'FALSE')

if auth_type and 'TRUE' == auth_type.upper() or 'YES' ==
auth_type.upper():
        filter = 'mschap:User-Name'
#else:
#       filter = 'Stripped-User-Name'
#print('\t\tfilter = "(uid=%%{%s:-%%{User-Name}})"' % filter)
@!@
filter = "(|(uid=%{mschap:User-Name:-%{User-
Name}})(macAddress=%{mschap:User-Name:-%{User-Name}}))"

                #  SASL parameters to use for user binds
                #
                #  When we're prompted by the SASL library, these
control
                #  the responses given.
                #
                #  Any of the config items below may be an attribute
ref
                #  or and expansion, so different SASL mechs, proxy IDs
                #  and realms may be used for different users.
                sasl {
                        # SASL mechanism
#                       mech = 'PLAIN'

                        # SASL authorisation identity to proxy.
#                       proxy = &User-Name
```

```
                        # SASL realm. Used for kerberos.
#                       realm = 'example.org'
                }

                #  Search scope, may be 'base', 'one', sub' or
'children'
#               scope = 'sub'

                #  Server side result sorting
                #
                #  A list of space delimited attributes to order the
result
                #  set by, if the filter matches multiple objects.
                #  Only the first result in the set will be processed.
                #
                #  If the attribute name is prefixed with a hyphen '-'
the
                #  sorting order will be reversed for that attribute.
                #
                #  If sort_by is set, and the server does not support
sorting
                #  the search will fail.
#               sort_by = '-uid'

                #  If this is undefined, anyone is authorised.
                #  If it is defined, the contents of this attribute
                #  determine whether or not the user is authorised
#               access_attribute = 'dialupAccess'

                #  Control whether the presence of 'access_attribute'
                #  allows access, or denys access.
                #
                #  If 'yes', and the access_attribute is present, or
                #  'no' and the access_attribute is absent then access
                #  will be allowed.
                #
                #  If 'yes', and the access_attribute is absent, or
                #  'no' and the access_attribute is present, then
                #  access will not be allowed.
                #
                #  If the value of the access_attribute is 'false', it
                #  will negate the result.
                #
                #  e.g.
                #    access_positive = yes
                #    access_attribute = userAccessAllowed
                #
                #  With an LDAP object containing:
                #    userAccessAllowed: false
                #
```

```
                #  Will result in the user being locked out.
#               access_positive = yes
        }

        #
        #  User membership checking.
        #
        group {
                #  Where to start searching in the tree for groups
                base_dn = "${..base_dn}"

                #  Filter for group objects, should match all available
                #  group objects a user might be a member of.
                filter = '(objectClass=posixGroup)'

                # Search scope, may be 'base', 'one', sub' or
'children'
#               scope = 'sub'

                #  Attribute that uniquely identifies a group.
                #  Is used when converting group DNs to group
                #  names.
#               name_attribute = cn

                #  Filter to find group objects a user is a member of.
                #  That is, group objects with attributes that
                #  identify members (the inverse of
membership_attribute).
#               membership_filter = "(|(member=%{control:Ldap-
UserDn})(memberUid=%{%{Stripped-User-Name}:-%{User-Name}}))"

                #  The attribute in user objects which contain the
names
                #  or DNs of groups a user is a member of.
                #
                #  Unless a conversion between group name and group DN
is
                #  needed, there's no requirement for the group objects
                #  referenced to actually exist.
                membership_attribute = 'memberOf'

                #  If cacheable_name or cacheable_dn are enabled,
                #  all group information for the user will be
                #  retrieved from the directory and written to LDAP-
Group
                #  attributes appropriate for the instance of rlm_ldap.
                #
                #  For group comparisons these attributes will be
checked
                #  instead of querying the LDAP directory directly.
                #
```

```
                #  This feature is intended to be used with rlm_cache.
                #
                #  If you wish to use this feature, you should enable
                #  the type that matches the format of your check items
                #  i.e. if your groups are specified as DNs then enable
                #  cacheable_dn else enable cacheable_name.
#               cacheable_name = 'no'
#               cacheable_dn = 'no'

                #  Override the normal cache attribute (<inst>-LDAP-
Group or
                #  LDAP-Group if using the default instance) and create
a
                #  custom attribute.  This can help if multiple module
instances
                #  are used in fail-over.
#               cache_attribute = 'LDAP-Cached-Membership'
        }

        #
        #  User profiles. RADIUS profile objects contain sets of
attributes
        #  to insert into the request. These attributes are mapped
using
        #  the same mapping scheme applied to user objects (the update
section above).
        #
        profile {
                #  Filter for RADIUS profile objects
#               filter = '(objectclass=radiusprofile)'

                #  The default profile.  This may be a DN or an
attribute
                #  reference.
                #  To get old v2.2.x style behavior, or to use the
                #  &User-Profile attribute to specify the default
profile,
                #  set this to &control:User-Profile.
#               default = 'cn=radprofile,dc=example,dc=org'

                #  The LDAP attribute containing profile DNs to apply
                #  in addition to the default profile above.  These are
                #  retrieved from the user object, at the same time as
the
                #  attributes from the update section, are are applied
                #  if authorization is successful.
                attribute = 'radiusProfileDn'
        }

        #
        #  Bulk load clients from the directory
```

```
        #
        client {
                #   Where to start searching in the tree for clients
                base_dn = "${..base_dn}"

                #
                #  Filter to match client objects
                #
                filter = '(objectClass=radiusClient)'

                # Search scope, may be 'base', 'one', 'sub' or
'children'
#               scope = 'sub'

                #
                #  Sets default values (not obtained from LDAP) for new
client entries
                #
                template {
#                       login                           = 'test'
#                       password                        = 'test'
#                       proto                           = tcp
#                       require_message_authenticator   = yes

                        # Uncomment to add a home_server with the same
                        # attributes as the client.
#                       coa_server {
#                               response_window = 2.0
#                       }
                }

                #
                #  Client attribute mappings are in the format:
                #      <client attribute> = <ldap attribute>
                #
                #  The following attributes are required:
                #    * ipaddr | ipv4addr | ipv6addr - Client IP
Address.
                #    * secret - RADIUS shared secret.
                #
                #  All other attributes usually supported in a client
                #  definition are also supported here.
                #
                #  Schemas are available in doc/schemas/ldap for
openldap and eDirectory
                #
                attribute {
                        ipaddr                          =
'radiusClientIdentifier'
                        secret                          =
'radiusClientSecret'
```

```
#                       shortname                       =
'radiusClientShortname'
#                       nas_type                        =
'radiusClientType'
#                       virtual_server                  =
'radiusClientVirtualServer'
#                       require_message_authenticator   =
'radiusClientRequireMa'
                }
        }

        #  Load clients on startup
#       read_clients = no

        #
        #  Modify user object on receiving Accounting-Request
        #

        #  Useful for recording things like the last time the user
logged
        #  in, or the Acct-Session-ID for CoA/DM.
        #
        #  LDAP modification items are in the format:
        #       <ldap attr> <op> <value>
        #
        #  Where:
        #       <ldap attr>:    The LDAP attribute to add modify or
delete.
        #       <op>:           One of the assignment operators:
        #                       (:=, +=, -=, ++).
        #                       Note: '=' is *not* supported.
        #       <value>:        The value to add modify or delete.
        #
        #  WARNING: If using the ':=' operator with a multi-valued LDAP
        #  attribute, all instances of the attribute will be removed
and
        #  replaced with a single attribute.
        accounting {
                reference = "%{tolower:type.%{Acct-Status-Type}}"

                type {
                        start {
                                update {
#                                       description := "Online at %S"
                                }
                        }

                        interim-update {
                                update {
#                                       description := "Last seen at
%S"
```

```
                                }
                        }

                        stop {
                                update {
#                                       description := "Offline at %S"
                                }
                        }
                }
        }

        #
        #  Post-Auth can modify LDAP objects too
        #
        post-auth {
                update {
#                       description := "Authenticated at %S"
                }
        }

        #
        #  LDAP connection-specific options.
        #
        #  These options set timeouts, keep-alive, etc. for the
connections.
        #
        options {
                #  Control under which situations aliases are followed.
                #  May be one of 'never', 'searching', 'finding' or
'always'
                #  default: libldap's default which is usually 'never'.
                #
                #  LDAP_OPT_DEREF is set to this value.
#               dereference = 'always'

                #
                #  The following two configuration items control
whether the
                #  server follows references returned by LDAP
directory.
                #  They are  mostly for Active Directory compatibility.
                #  If you set these to 'no', then searches will likely
return
                #  'operations error', instead of a useful result.
                #
                chase_referrals = yes
                rebind = yes

                #  Seconds to wait for LDAP query to finish. default:
20
                res_timeout = 10
```

```
                #  Seconds LDAP server has to process the query
(server-side
                #  time limit). default: 20
                #
                #  LDAP_OPT_TIMELIMIT is set to this value.
                srv_timelimit = 3

                #  Seconds to wait for response of the server. (network
                #  failures) default: 10
                #
                #  LDAP_OPT_NETWORK_TIMEOUT is set to this value.
                net_timeout = 1

                #  LDAP_OPT_X_KEEPALIVE_IDLE
                idle = 60

                #  LDAP_OPT_X_KEEPALIVE_PROBES
                probes = 3

                #  LDAP_OPT_X_KEEPALIVE_INTERVAL
                interval = 3

                #  ldap_debug: debug flag for LDAP SDK
                #  (see OpenLDAP documentation).  Set this to enable
                #  huge amounts of LDAP debugging on the screen.
                #  You should only use this if you are an LDAP expert.
                #
                #       default: 0x0000 (no debugging messages)
                #       Example:(LDAP_DEBUG_FILTER+LDAP_DEBUG_CONNS)
                ldap_debug = 0x0028
        }

        #
        #  This subsection configures the tls related items
        #  that control how FreeRADIUS connects to an LDAP
        #  server.  It contains all of the 'tls_*' configuration
        #  entries used in older versions of FreeRADIUS.  Those
        #  configuration entries can still be used, but we recommend
        #  using these.
        #
        tls {
                # Set this to 'yes' to use TLS encrypted connections
                # to the LDAP database by using the StartTLS extended
                # operation.
                #
                # The StartTLS operation is supposed to be
                # used with normal ldap connections instead of
                # using ldaps (port 636) connections
#               start_tls = yes
@!@
```

```
print('\t\tstart_tls = %s' %
configRegistry.get('freeradius/conf/starttls', 'no'))
@!@

#               ca_file = ${certdir}/cacert.pem

#               ca_path = ${certdir}
#               certificate_file = /path/to/radius.crt
#               private_key_file = /path/to/radius.key
#               random_file = /dev/urandom

                #  Certificate Verification requirements.  Can be:
                #    'never' (do not even bother trying)
                #    'allow' (try, but don't fail if the certificate
                #                cannot be verified)
                #    'demand' (fail if the certificate does not verify)
                #    'hard'  (similar to 'demand' but fails if TLS
                #             cannot negotiate)
                #
                #  The default is libldap's default, which varies based
                #  on the contents of ldap.conf.

#               require_cert    = 'demand'
        }

        #  As of version 3.0, the 'pool' section has replaced the
        #  following configuration items:
        #
        #  ldap_connections_number

        #  The connection pool is new for 3.0, and will be used in many
        #  modules, for all kinds of connection-related activity.
        #
        #  When the server is not threaded, the connection pool
        #  limits are ignored, and only one connection is used.
        pool {
                #  Connections to create during module instantiation.
                #  If the server cannot create specified number of
                #  connections during instantiation it will exit.
                #  Set to 0 to allow the server to start without the
                #  directory being available.
                start = ${thread[pool].start_servers}

                #  Minimum number of connections to keep open
                min = ${thread[pool].min_spare_servers}

                #  Maximum number of connections
                #
                #  If these connections are all in use and a new one
                #  is requested, the request will NOT get a connection.
                #
```

```
                #  Setting 'max' to LESS than the number of threads
means
                #  that some threads may starve, and you will see
errors
                #  like 'No connections available and at max connection
limit'
                #
                #  Setting 'max' to MORE than the number of threads
means
                #  that there are more connections than necessary.
                max = ${thread[pool].max_servers}

                #  Spare connections to be left idle
                #
                #  NOTE: Idle connections WILL be closed if
"idle_timeout"
                #  is set.  This should be less than or equal to "max"
above.
                spare = ${thread[pool].max_spare_servers}

                #  Number of uses before the connection is closed
                #
                #  0 means "infinite"
                uses = 0

                #  The number of seconds to wait after the server tries
                #  to open a connection, and fails.  During this time,
                #  no new connections will be opened.
                retry_delay = 30

                #  The lifetime (in seconds) of the connection
                lifetime = 0

                #  Idle timeout (in seconds).  A connection which is
                #  unused for this length of time will be closed.
                idle_timeout = 60

                #  NOTE: All configuration settings are enforced.  If a
                #  connection is closed because of 'idle_timeout',
                #  'uses', or 'lifetime', then the total number of
                #  connections MAY fall below 'min'.  When that
                #  happens, it will open a new connection.  It will
                #  also log a WARNING message.
                #
                #  The solution is to either lower the 'min'
connections,
                #  or increase lifetime/idle_timeout.
        }
}
```

ldap_orig_UCS5

```
@%@UCRWARNING=# @%@
# -*- text -*-
#
#  $Id: 4b7e4585c029b8617aa7b9169a42bf50a5ec4938 $

#
#  Lightweight Directory Access Protocol (LDAP)
#
ldap {
        #  Note that this needs to match the name(s) in the LDAP server
        #  certificate, if you're using ldaps.  See OpenLDAP
documentation
        #  for the behavioral semantics of specifying more than one
host.
        #
        #  Depending on the libldap in use, server may be an LDAP URI.
        #  In the case of OpenLDAP this allows additional the following
        #  additional schemes:
        #  - ldaps:// (LDAP over SSL)
        #  - ldapi:// (LDAP over Unix socket)
        #  - ldapc:// (Connectionless LDAP)
        server = "@%@ldap/server/name@%@"
#       server = 'ldap.rrdns.example.org'
#       server = 'ldap.rrdns.example.org'

        #  Port to connect on, defaults to 389, will be ignored for
LDAP URIs.
#       port = 389
@!@
print('\tport = "%s"' % (configRegistry.get('ldap/server/port',
'7389'), ))
@!@

        #  Administrator account for searching and possibly modifying.
        #  If using SASL + KRB5 these should be commented out.
#       identity = 'cn=admin,dc=example,dc=org'
        identity = "@%@ldap/hostdn@%@"
#       password = mypass
@!@
with open('/etc/machine.secret', 'r') as credfile:
        print("\tpassword = \"%s\"" % credfile.readline().rstrip("\n"))
@!@

        #  Unless overridden in another section, the dn from which all
        #  searches will start from.
#       base_dn = 'dc=example,dc=org'
        base_dn = "@%@ldap/base@%@"

        #
```

```
        #  SASL parameters to use for admin binds
        #
        #  When we're prompted by the SASL library, these control
        #  the responses given, as well as the identity and password
        #  directives above.
        #
        #  If any directive is commented out, a NULL response will be
        #  provided to cyrus-sasl.
        #
        #  Unfortunately the only way to control Keberos here is
through
        #  environmental variables, as cyrus-sasl provides no API to
        #  set the krb5 config directly.
        #
        #  Full documentation for MIT krb5 can be found here:
        #
        #
http://web.mit.edu/kerberos/krb5-devel/doc/admin/env_variables.html
        #
        #  At a minimum you probably want to set KRB5_CLIENT_KTNAME.
        #
        sasl {
                # SASL mechanism
#               mech = 'PLAIN'

                # SASL authorisation identity to proxy.
#               proxy = 'autz_id'

                # SASL realm. Used for kerberos.
#               realm = 'example.org'
        }

        #
        #  Generic valuepair attribute
        #

        #  If set, this will attribute will be retrieved in addition to
any
        #  mapped attributes.
        #
        #  Values should be in the format:
        #       <radius attr> <op> <value>
        #
        #  Where:
        #       <radius attr>:  Is the attribute you wish to create
        #                       with any valid list and request
qualifiers.
        #       <op>:           Is any assignment operator (=, :=, +=,
-=).
        #       <value>:        Is the value to parse into the new
valuepair.
```

```
        #                       If the value is wrapped in double
quotes it
        #                       will be xlat expanded.
#       valuepair_attribute = 'radiusAttribute'

        #
        #  Mapping of LDAP directory attributes to RADIUS dictionary
attributes.
        #

        #  WARNING: Although this format is almost identical to the
unlang
        #  update section format, it does *NOT* mean that you can use
other
        #  unlang constructs in module configuration files.
        #
        #  Configuration items are in the format:
        #       <radius attr> <op> <ldap attr>
        #
        #  Where:
        #       <radius attr>:  Is the destination RADIUS attribute
        #                       with any valid list and request
qualifiers.
        #       <op>:           Is any assignment attribute (=, :=, +=,
-=).
        #       <ldap attr>:    Is the attribute associated with user
or
        #                       profile objects in the LDAP directory.
        #                       If the attribute name is wrapped in
double
        #                       quotes it will be xlat expanded.
        #
        #  Request and list qualifiers may also be placed after the
'update'
        #  section name to set defaults destination requests/lists
        #  for unqualified RADIUS attributes.
        #
        #  Note: LDAP attribute names should be single quoted unless
you want
        #  the name value to be derived from an xlat expansion, or an
        #  attribute ref.
        update {
                control:Password-With-Header    += 'userPassword'
#               control:NT-Password             := 'ntPassword'
#               reply:Reply-Message             := 'radiusReplyMessage'
#               reply:Tunnel-Type               := 'radiusTunnelType'
#               reply:Tunnel-Medium-Type        :=
'radiusTunnelMediumType'
#               reply:Tunnel-Private-Group-ID   :=
'radiusTunnelPrivategroupId'
```

```
                #  Where only a list is specified as the RADIUS
attribute,
                #  the value of the LDAP attribute is parsed as a
valuepair
                #  in the same format as the 'valuepair_attribute'
(above).
                control:                        +=
'radiusControlAttribute'
                request:                        +=
'radiusRequestAttribute'
                reply:                          +=
'radiusReplyAttribute'
@!@
auth_type = configRegistry.get('freeradius/conf/auth-type/mschap',
'FALSE')

if auth_type and 'TRUE' == auth_type.upper() or 'YES' ==
auth_type.upper():
        print('control:LM-Password\t\t:=\tsambaLMPassword')
        print('control:NT-Password\t\t:=\tsambaNTPassword')
else:
        print('control:LM-Password\t\t:=\tlmPassword')
        print('control:NT-Password\t\t:=\tntPassword')
        print('control:LM-Password\t\t:=\tsambaLmPassword')
        print('control:NT-Password\t\t:=\tsambaNtPassword')
@!@
        }

        #  Set to yes if you have eDirectory and want to use the
universal
        #  password mechanism.
#       edir = no

        #  Set to yes if you want to bind as the user after retrieving
the
        #  Cleartext-Password. This will consume the login grace, and
        #  verify user authorization.
#       edir_autz = no

        #  Note: set_auth_type was removed in v3.x.x
        #  Equivalent functionality can be achieved by adding the
following
        #  stanza to the authorize {} section of your virtual server.
        #
        #    ldap
        #    if ((ok || updated) && User-Password) {
        #        update {
        #            control:Auth-Type := ldap
        #        }
        #    }
```

```
        #
        #  User object identification.
        #
        user {
                #  Where to start searching in the tree for users
                base_dn = "${..base_dn}"

                #  Filter for user objects, should be specific enough
                #  to identify a single user object.
                #
                #  For Active Directory, you should use
                #  "samaccountname=" instead of "uid="
                #
#               filter = "(uid=%{%{Stripped-User-Name}:-%{User-Name}})"
@!@
auth_type = configRegistry.get('freeradius/conf/auth-type/mschap',
'FALSE')

if auth_type and 'TRUE' == auth_type.upper() or 'YES' ==
auth_type.upper():
        filter = 'mschap:User-Name'
else:
        filter = 'Stripped-User-Name'
print('\t\tfilter = "(uid=%%{%s:-%%{User-Name}})"' % filter)
@!@

                #  SASL parameters to use for user binds
                #
                #  When we're prompted by the SASL library, these
control
                #  the responses given.
                #
                #  Any of the config items below may be an attribute
ref
                #  or and expansion, so different SASL mechs, proxy IDs
                #  and realms may be used for different users.
                sasl {
                        # SASL mechanism
#                       mech = 'PLAIN'

                        # SASL authorisation identity to proxy.
#                       proxy = &User-Name

                        # SASL realm. Used for kerberos.
#                       realm = 'example.org'
                }

                #  Search scope, may be 'base', 'one', sub' or
'children'
#               scope = 'sub'
```

```
                #  Server side result sorting
                #
                #  A list of space delimited attributes to order the
result
                #  set by, if the filter matches multiple objects.
                #  Only the first result in the set will be processed.
                #
                #  If the attribute name is prefixed with a hyphen '-'
the
                #  sorting order will be reversed for that attribute.
                #
                #  If sort_by is set, and the server does not support
sorting
                #  the search will fail.
#               sort_by = '-uid'

                #  If this is undefined, anyone is authorised.
                #  If it is defined, the contents of this attribute
                #  determine whether or not the user is authorised
#               access_attribute = 'dialupAccess'

                #  Control whether the presence of 'access_attribute'
                #  allows access, or denys access.
                #
                #  If 'yes', and the access_attribute is present, or
                #  'no' and the access_attribute is absent then access
                #  will be allowed.
                #
                #  If 'yes', and the access_attribute is absent, or
                #  'no' and the access_attribute is present, then
                #  access will not be allowed.
                #
                #  If the value of the access_attribute is 'false', it
                #  will negate the result.
                #
                #  e.g.
                #    access_positive = yes
                #    access_attribute = userAccessAllowed
                #
                #  With an LDAP object containing:
                #    userAccessAllowed: false
                #
                #  Will result in the user being locked out.
#               access_positive = yes
        }

        #
        #  User membership checking.
        #
        group {
                #  Where to start searching in the tree for groups
```

```
                base_dn = "${..base_dn}"

                #  Filter for group objects, should match all available
                #  group objects a user might be a member of.
                filter = '(objectClass=posixGroup)'

                # Search scope, may be 'base', 'one', sub' or
'children'
#               scope = 'sub'

                #  Attribute that uniquely identifies a group.
                #  Is used when converting group DNs to group
                #  names.
#               name_attribute = cn

                #  Filter to find group objects a user is a member of.
                #  That is, group objects with attributes that
                #  identify members (the inverse of
membership_attribute).
#               membership_filter = "(|(member=%{control:Ldap-
UserDn})(memberUid=%{%{Stripped-User-Name}:-%{User-Name}}))"

                #  The attribute in user objects which contain the
names
                #  or DNs of groups a user is a member of.
                #
                #  Unless a conversion between group name and group DN
is
                #  needed, there's no requirement for the group objects
                #  referenced to actually exist.
                membership_attribute = 'memberOf'

                #  If cacheable_name or cacheable_dn are enabled,
                #  all group information for the user will be
                #  retrieved from the directory and written to LDAP-
Group
                #  attributes appropriate for the instance of rlm_ldap.
                #
                #  For group comparisons these attributes will be
checked
                #  instead of querying the LDAP directory directly.
                #
                #  This feature is intended to be used with rlm_cache.
                #
                #  If you wish to use this feature, you should enable
                #  the type that matches the format of your check items
                #  i.e. if your groups are specified as DNs then enable
                #  cacheable_dn else enable cacheable_name.
#               cacheable_name = 'no'
#               cacheable_dn = 'no'
```

```
                #  Override the normal cache attribute (<inst>-LDAP-
Group or
                #  LDAP-Group if using the default instance) and create
a
                #  custom attribute.  This can help if multiple module
instances
                #  are used in fail-over.
#               cache_attribute = 'LDAP-Cached-Membership'
        }

        #
        #  User profiles. RADIUS profile objects contain sets of
attributes
        #  to insert into the request. These attributes are mapped
using
        #  the same mapping scheme applied to user objects (the update
section above).
        #
        profile {
                #  Filter for RADIUS profile objects
#               filter = '(objectclass=radiusprofile)'

                #  The default profile.  This may be a DN or an
attribute
                #  reference.
                #  To get old v2.2.x style behavior, or to use the
                #  &User-Profile attribute to specify the default
profile,
                #  set this to &control:User-Profile.
#               default = 'cn=radprofile,dc=example,dc=org'

                #  The LDAP attribute containing profile DNs to apply
                #  in addition to the default profile above.  These are
                #  retrieved from the user object, at the same time as
the
                #  attributes from the update section, are are applied
                #  if authorization is successful.
#               attribute = 'radiusProfileDn'
        }

        #
        #  Bulk load clients from the directory
        #
        client {
                #   Where to start searching in the tree for clients
                base_dn = "${..base_dn}"

                #
                #  Filter to match client objects
                #
                filter = '(objectClass=radiusClient)'
```

```
                # Search scope, may be 'base', 'one', 'sub' or
'children'
#               scope = 'sub'

                #
                #  Sets default values (not obtained from LDAP) for new
client entries
                #
                template {
#                       login                           = 'test'
#                       password                        = 'test'
#                       proto                           = tcp
#                       require_message_authenticator   = yes

                        # Uncomment to add a home_server with the same
                        # attributes as the client.
#                       coa_server {
#                               response_window = 2.0
#                       }
                }

                #
                #  Client attribute mappings are in the format:
                #      <client attribute> = <ldap attribute>
                #
                #  The following attributes are required:
                #    * ipaddr | ipv4addr | ipv6addr - Client IP
Address.
                #    * secret - RADIUS shared secret.
                #
                #  All other attributes usually supported in a client
                #  definition are also supported here.
                #
                #  Schemas are available in doc/schemas/ldap for
openldap and eDirectory
                #
                attribute {
                        ipaddr                          =
'radiusClientIdentifier'
                        secret                          =
'radiusClientSecret'
#                       shortname                       =
'radiusClientShortname'
#                       nas_type                        =
'radiusClientType'
#                       virtual_server                  =
'radiusClientVirtualServer'
#                       require_message_authenticator   =
'radiusClientRequireMa'
                }
```

```
        }

        #  Load clients on startup
#       read_clients = no

        #
        #  Modify user object on receiving Accounting-Request
        #

        #  Useful for recording things like the last time the user
logged
        #  in, or the Acct-Session-ID for CoA/DM.
        #
        #  LDAP modification items are in the format:
        #       <ldap attr> <op> <value>
        #
        #  Where:
        #       <ldap attr>:    The LDAP attribute to add modify or
delete.
        #       <op>:           One of the assignment operators:
        #                       (:=, +=, -=, ++).
        #                       Note: '=' is *not* supported.
        #       <value>:        The value to add modify or delete.
        #
        #  WARNING: If using the ':=' operator with a multi-valued LDAP
        #  attribute, all instances of the attribute will be removed
and
        #  replaced with a single attribute.
        accounting {
                reference = "%{tolower:type.%{Acct-Status-Type}}"

                type {
                        start {
                                update {
#                                       description := "Online at %S"
                                }
                        }

                        interim-update {
                                update {
#                                       description := "Last seen at
%S"
                                }
                        }

                        stop {
                                update {
#                                       description := "Offline at %S"
                                }
                        }
                }
```

```
        }

        #
        #  Post-Auth can modify LDAP objects too
        #
        post-auth {
                update {
#                       description := "Authenticated at %S"
                }
        }

        #
        #  LDAP connection-specific options.
        #
        #  These options set timeouts, keep-alive, etc. for the
connections.
        #
        options {
                #  Control under which situations aliases are followed.
                #  May be one of 'never', 'searching', 'finding' or
'always'
                #  default: libldap's default which is usually 'never'.
                #
                #  LDAP_OPT_DEREF is set to this value.
#               dereference = 'always'

                #
                #  The following two configuration items control
whether the
                #  server follows references returned by LDAP
directory.
                #  They are  mostly for Active Directory compatibility.
                #  If you set these to 'no', then searches will likely
return
                #  'operations error', instead of a useful result.
                #
                chase_referrals = yes
                rebind = yes

                #  Seconds to wait for LDAP query to finish. default:
20
                res_timeout = 10

                #  Seconds LDAP server has to process the query
(server-side
                #  time limit). default: 20
                #
                #  LDAP_OPT_TIMELIMIT is set to this value.
                srv_timelimit = 3

                #  Seconds to wait for response of the server. (network
```

```
                #  failures) default: 10
                #
                #  LDAP_OPT_NETWORK_TIMEOUT is set to this value.
                net_timeout = 1

                #  LDAP_OPT_X_KEEPALIVE_IDLE
                idle = 60

                #  LDAP_OPT_X_KEEPALIVE_PROBES
                probes = 3

                #  LDAP_OPT_X_KEEPALIVE_INTERVAL
                interval = 3

                #  ldap_debug: debug flag for LDAP SDK
                #  (see OpenLDAP documentation).  Set this to enable
                #  huge amounts of LDAP debugging on the screen.
                #  You should only use this if you are an LDAP expert.
                #
                #       default: 0x0000 (no debugging messages)
                #       Example:(LDAP_DEBUG_FILTER+LDAP_DEBUG_CONNS)
                ldap_debug = 0x0028
        }

        #
        #  This subsection configures the tls related items
        #  that control how FreeRADIUS connects to an LDAP
        #  server.  It contains all of the 'tls_*' configuration
        #  entries used in older versions of FreeRADIUS.  Those
        #  configuration entries can still be used, but we recommend
        #  using these.
        #
        tls {
                # Set this to 'yes' to use TLS encrypted connections
                # to the LDAP database by using the StartTLS extended
                # operation.
                #
                # The StartTLS operation is supposed to be
                # used with normal ldap connections instead of
                # using ldaps (port 636) connections
#               start_tls = yes
@!@
print('\t\tstart_tls = %s' %
configRegistry.get('freeradius/conf/starttls', 'no'))
@!@

#               ca_file = ${certdir}/cacert.pem

#               ca_path = ${certdir}
#               certificate_file = /path/to/radius.crt
#               private_key_file = /path/to/radius.key
```

```
#               random_file = /dev/urandom

                #  Certificate Verification requirements.  Can be:
                #    'never' (do not even bother trying)
                #    'allow' (try, but don't fail if the certificate
                #               cannot be verified)
                #    'demand' (fail if the certificate does not verify)
                #    'hard'  (similar to 'demand' but fails if TLS
                #             cannot negotiate)
                #
                #  The default is libldap's default, which varies based
                #  on the contents of ldap.conf.

#               require_cert    = 'demand'
        }

        #  As of version 3.0, the 'pool' section has replaced the
        #  following configuration items:
        #
        #  ldap_connections_number

        #  The connection pool is new for 3.0, and will be used in many
        #  modules, for all kinds of connection-related activity.
        #
        #  When the server is not threaded, the connection pool
        #  limits are ignored, and only one connection is used.
        pool {
                #  Connections to create during module instantiation.
                #  If the server cannot create specified number of
                #  connections during instantiation it will exit.
                #  Set to 0 to allow the server to start without the
                #  directory being available.
                start = ${thread[pool].start_servers}

                #  Minimum number of connections to keep open
                min = ${thread[pool].min_spare_servers}

                #  Maximum number of connections
                #
                #  If these connections are all in use and a new one
                #  is requested, the request will NOT get a connection.
                #
                #  Setting 'max' to LESS than the number of threads
means
                #  that some threads may starve, and you will see
errors
                #  like 'No connections available and at max connection
limit'
                #
                #  Setting 'max' to MORE than the number of threads
means
```

```
                #  that there are more connections than necessary.
                max = ${thread[pool].max_servers}

                #  Spare connections to be left idle
                #
                #  NOTE: Idle connections WILL be closed if
"idle_timeout"
                #  is set.  This should be less than or equal to "max"
above.
                spare = ${thread[pool].max_spare_servers}

                #  Number of uses before the connection is closed
                #
                #  0 means "infinite"
                uses = 0

                #  The number of seconds to wait after the server tries
                #  to open a connection, and fails.  During this time,
                #  no new connections will be opened.
                retry_delay = 30

                #  The lifetime (in seconds) of the connection
                lifetime = 0

                #  Idle timeout (in seconds).  A connection which is
                #  unused for this length of time will be closed.
                idle_timeout = 60

                #  NOTE: All configuration settings are enforced.  If a
                #  connection is closed because of 'idle_timeout',
                #  'uses', or 'lifetime', then the total number of
                #  connections MAY fall below 'min'.  When that
                #  happens, it will open a new connection.  It will
                #  also log a WARNING message.
                #
                #  The solution is to either lower the 'min'
connections,
                #  or increase lifetime/idle_timeout.
        }
}
```