Maja Górecka-Wolniewicz
UCI, UMK

INSTALACJA I KONFIGURACJA SERWERA FREERADIUS

Projekt FreeRADIUS

Jest to jedna z bardziej popularnych implementacji niekomercyjnych, powszechnie wykorzystywana w projekcie eduroam, zalecana w polskim projekcie eduroam.
Stroną projektu jest http://www.freeradius.org. Aktualna, opisywana w tym dokumencie wersja to 1.1.0.
Oprogramowanie FreeRADIUS cechują:

- duże możliwości (różnorodność modułów),
- elastyczność w konfiguracji.

Wymagane oprogramowanie:

- pakiet OpenSSL - nie jest potrzebny, gdy serwer pełni wyłącznie funkcję proxy,
- pakiet OpenLDAP - jeśli korzystamy z usługi LDAP do uwierzytelniania/autoryzacji (i nie korzystamy z komercyjnego oprogramowania LDAP),
- oprogramowanie dodatkowe, zgodnie z potrzebami, np. MySQL, Postgres etc.

Obsługa wyrażeń regułowych w realmach

Domyślnie FreeRADIUS pozwala na dopasowywanie wartości realm na podstawie porównania: wartość taka sama lub różna.
Patch 'Enable regular expressions matching in realms', udostępniany na stronie:
http://projects.nuschkys.net/patches
pozwala na podanie w polu wartości wyrażenia regułowego, do którego ma zostać dopasowany realm.
Zastosowanie patcha (w katalogu dystrybucji) odbywa się przez uruchomienie polecenia:

```
patch -p1 < rozpakowany_plik_z_poprawką
```

Instalacja oprogramowania FreeRADIUS

Na początek należy rozwinąć źródła oprogramowania pobranego ze strony www.freeradius.org.
Następnie wykonujemy polecenia:

```
cd freeradius-1.1.0
./configure -prefix=/opt/freeradius
```

Gdy chcemy użyć własnej instalacji LDAP:

```
--with-rlm-ldap-lib=/opt/ldap/lib
--with-rlm-ldap-include=/opt/ldap/include
```

Gdy chcemy użyć własnej instalacji OpenSSL:

```
--with-ssl-lib=/opt/ssl/lib
--with-ssl-include=/opt/ssl/include
```

Uwaga:
Jeśli używamy własnej instalacji OpenSSL-a, aby zagwarantować, że programy i biblioteki są dolinkowane do właściwych bibliotek OpenSSL najlepiej, przed wywołaniem `configure` ustawić zmienne środowiskowe `CPPFLAGS` i `LDFLAGS`, np.:

```
CPPFLAGS=-I/opt/ldap/include -I/opt/ssl/include
LDFLAGS=-L/opt/ldap/lib -L/opt/ssl/lib -Xlinker \
     -R/opt/ldap/lib:/opt/ssl/lib
```

Inne przykładowe ustawienia na etapie kompilacji :

```
--with-experimental-modules
--without-rlm-perl --without-rlm-sql
```

Kompilacja:
`make`

Instalacja (na ogół z uprawnieniami root):
`make install`

Konfiguracja oprogramowania FreeRADIUS

Nasze założenia konfiguracyjne są następujące:

- zmierzamy do uruchomienia serwera FreeRADIUS instytucji (poziom 3 w hierarchii eduroam), co oznacza, że jest wymagana możliwość kontaktowania się z serwerem ogólnopolskim, ew. z serwerami poziomu 4;
- realizujemy uwierzytelnianie i autoryzację użytkowników lokalnych (danej instytucji, w zdefiniowanym przez instytucję zakresie), zgodnie z przyjętymi na jej terenie zasadami uwierzytelniania (baza LDAP, pliki Unix itp.).

Pliki konfiguracyjne FreeRADIUS-a znajdują się w katalogu `${katalog_instalacji}/etc/raddb` instalacji pakietu FreeRADIUS. Najistotniejsze pliki konfiguracyjne to:

- `radiusd.conf`
- `clients.conf`
- `proxy.conf`
- `users`
- `huntgroups`

Plik `clients.conf`

Plik ten służy do specyfikacji adresów lub nazw urządzeń (Access Pointy, Switche 802.1x) oraz serwerów RADIUS, z którymi konfigurowany serwer RADIUS będzie się komunikował.
Postać definicji jest następująca:

```
client <nazwa | adres-IP | podsieć> {
<atrybut> = <wartość>
}
```

Na przykład:

```
client 192.168.1.120 {
 secret = tajny_klucz_RADIUS-AP
 shorname = campus1dev120
}
```

W atrybutach dla danego klienta można dodać `nastype = <słownikowa_wartość>`
`nastype` mówi, jakiej metody użyć w skrypcie `checkrad` (program udostępniany w dystrybucji FreeRADIUS) do sprawdzenia jednoczesnego zalogowania użytkownika.

Uwagi:

- należy dbać o unikatowość kluczy;
- klucz tajny powinien mieć min. 16 znaków;
- zaleca się różnicowanie kluczy wspólnych, nie powinno definiować się podsieci AP-ów z jednym hasłem wspólnym;
- klientami są również serwery RADIUS – te, z którymi dany serwer kontaktuje się należy zdefiniować w pliku `clients.conf` i ustalić wspólny klucz do komunikacji;
- w testach przydaje się klient loopback (127.0.0.1).

Plik `huntgroups`

Służy do definicji grup urządzeń NAS.
Postać definicji:

```
  aptest NAS-IP-Address == 192.168.1.200
  aptest NAS-IP-Address == 192.168.1.201
```

lub (bardziej złożona)

```
 aptest NAS-IP-Address == 192.168.1.201, NAS-Port-Id == 0-3
          User-Name = user1,
          User-Name = user2
```

Z definicji zastosowanych w pliku `huntgroups` korzysta plik `users`. Można w nim zdefiniować kryterium dopasowania, np. `Hundgroup-Name == "aptest"`

Plik `proxy.conf`

Za pomocą tego pliku odbywa się konfiguracja realmów obsługiwanych przez serwer.
Deklaracja lokalnej obsługi realmu ma postać:

```
realm a.b {
 type = radius
 authhost = LOCAL
 accthost = LOCAL
}
```

Deklaracja wskazująca, że realm c.d jest obsługiwany przez zdalny serwer:

```
realm c.d {
 type = radius
 authhost = 10.1.1.1:1812
 accthost = 10.1.1.1:1813
 secret = a2s3d4f5g6
 nostrip
}
```

`secret` – wskazuje hasło wspólne stosowane przez dany serwer w komunikacji z danym klientem
`nostrip` – opcja powodująca, że nazwa sieciowa użytkownika nie jest pozbawiana nazwy domenowej (realmu) przed wysłaniem zlecenia do serwera (domyślnie realm jest obcinany), stosowanie `nostrip` jest zlecane w projekcie eduroam – do serwerów proxy należy przekazywać pełne nazwy sieciowe użytkowników.
Jeśli zależy nam na korzystaniu z wyrażeń regułowych w celu dopasowania realmu, niezbędne jest zastosowanie opisanego wyżej patcha. Następnie w bloku definicji realmu zamieszczamy atrybut
`regex = wyrażenie_regułowe`

Na przykład:

```
realm e.f {
 regex = "@.*\.e\.f$"
 type = radius
 authhost = 10.1.1.1:1812
 accthost = 10.1.1.1:1813
 secret = a2s3d4f5g6
 nostrip
}
```

Dla danego realmu można zbudować więcej niż jedną definicję obsługi. W ten sposób realizowana jest redundancja serwerów RADIUS. Na przykład, definicja:

```
realm c.d {
 type = radius
 authhost = radius1.my.org:1812
 accthost = radius1.my.org:1813
 secret = a2s3d4f5g6
 nostrip
}
realm c.d {
 type = radius
 authhost = radius2.my.org:1812
 accthost = radius2.my.org:1813
 secret = a2s3d4f5g6
 nostrip
}
```

powoduje, że jeśli serwer RADIUS o nazwie `radius1.my.org` nie jest dostępny, to realm obsłuży serwer `radius2.my.org` (jeśli jest dostępny).
Drugim wykorzystaniem możliwości wielokrotnej definicji tego samego ralmu w pliku **`proxy.conf`** jest obsługa tzw. load balancingu, tj. równomiernego rozłożenia zleceń do serwerów. Jeśli dopiszemy przy definicji realmu:

```
ldflag = round_robin
```

to wszystkie aktywne realmy dla danej nazwy biorą udział w procesie wyboru docelowego serwera – takie podejście zalecane jest w odniesieniu do realmów obsługiwanych przez serwery krajowe.
Domyślnie, gdy nie zadeklarowano `ldflags`, przyjmowana jest wartość

```
ldflag = fail_over
```

co oznacza, definicje realmów przeglądane są w kolejności ich deklaracji, pierwszy aktywny serwer obsługuje realm.
Sekcja `proxy_server` w pliku `proxy.conf` kontroluje działanie serwera w komunikacji proxy. M.in. możliwe jest ustalenie następujących parametrów:

- `synchronous` – jeśli ma być aktywna funkcjonalność *fail-over*, to `synchronous = no`; wartość `yes` oznacza, że zlecenia będą automatycznie ponownie wysyłane do serwera proxy, natychmiast gdy NAS ponowi wysłanie;
- `retry_delay` (5) – czas (w sekundach) oczekiwania na odpowiedź, przed ponownym przesłaniem zlecenia proxy;
- `retry_count` (3) – ile razy ponawiać wysłanie zlecenia proxy przed odpowiedzią *reject*;
- `dead_time` (120) – ile sekund odczekać przed sprawdzeniem, czy serwer proxy stał się dostępny;
- `default_fallback` – obsługa alternatywna wg dyrektywy DEFAULT (gdy `yes`)
- `post_proxy_authorize` - sprawdź sekcję autoryzacji (na ogół m.in. plik `users`) po powrocie z proxy.

Plik `users`

Plik konfiguracyjny `users` jest używany przez moduł `files`. Zastosowania modułu `files` to:

- autoryzacja użytkownika (`authorize`),
- rozliczenie (`preacct`),
- obsługa użytkowników przed realizacją funkcji proxy (`pre-proxy`) .

Domyślne nazwy plików konfiguracji są zamieszczone w sekcji `files`:

```
usersfile = ${confdir}/users
acctusersfile = ${confdir}/acct_users
preproxy_usersfile = ${confdir}/preproxy_users
```

W pliku `users` umieszcza się zestaw dyrektyw konfiguracyjnych używanych przez moduł `files` w celu podjęcia decyzji o sposobie autoryzacji i uwierzytelnienia użytkownika. Format pliku pliku jest następujący:

```
nazwa <lista_sprawdzanych_elementów (check items)> [, ]\
  <lista_elem_konfiguracyjnych>
TAB <lista_elementów_odpowiedzi (reply items)>
```

`nazwa` – nazwa użytkownika lub słowo DEFAULT (dowolny użytkownik)

Uwagi:

- wszystkie elementy sprawdzane MUSZĄ być umieszczone w jednym wierszu, za nazwą użytkownika;
- separatorem elementów sprawdzanych jest przecinek;
- w pierwszym wierszu, w ramach listy elementów sprawdzanych MOŻNA wskazać element konfiguracyjny, np. Auth-Type i/lub Autz-Type
- elementy odpowiedzi MOGĄ być umieszczane w wielu wierszach, w tym przypadku poprzedni wiersz (zawierający element odpowiedzi) musi kończyć się przecinkiem;
- przetwarzanie pliku `users` jest realizowane sekwencyjnie od początku pliku;
- po dopasowaniu użytkownika i elementów sprawdzanych ustalane są elementy odpowiedzi (wg deklaracji);
- domyślnie po dopasowaniu przetwarzanie `users` kończy się;
- element specjalny: `Fall-Through = Yes` oznacza "przetwarzaj dalej".

Dozwolone postaci definicji atrybut – wartość:

1. forma `atrybut = wartość`
   NIE MOŻE wystąpić na liście sprawdzanych elementów
   MOŻE wystąpić jako element konfiguracyjny (Auth-Type) - ustala wartość atrybutu, tylko gdy nie była ustalona na liście odpowiedzi oznacza dodanie wartości danego atrybutu, ale tylko, gdy atrybut nie ma dotychczas wartości
2. forma `atrybut := wartość`
   na liście sprawdzanych elementów oznacza "zawsze dopasowany" zastępuje wartość atrybutu konfiguracyjnego
   zastępuje wartość atrybutu konfiguracyjnego na liście odpowiedzi
3. forma `atrybut += wartość`
   na liście sprawdzanych elementów oznacza "zawsze dopasowany"
   na liście konfiguracji i odpowiedzi oznacza dodanie wartości danego atrybutu
4. forma `atrybut == wartość`
   na liście sprawdzanych elementów sprawdza, czy atrybut istnieje w zleceniu i czy wartość jest równa podanej
   NIE MOŻE wystąpić na liście odpowiedzi
5. forma `atrybut != wartość`
   na liście sprawdzanych elementów sprawdza, czy atrybut istnieje w zleceniu i wartość jest różna od podanej
   NIE MOŻE wystąpić na liście odpowiedzi
6. forma `atrybut > wartość`
   na liście sprawdzanych elementów sprawdza, czy wartość atrybutu w zleceniu jest większa od podanej
   NIE MOŻE wystąpić na liście odpowiedzi
7. forma `atrybut >= wartość`
   na liście sprawdzanych elementów sprawdza, czy wartość atrybutu w zleceniu jest większa lub równa podanej
   NIE MOŻE wystąpić na liście odpowiedzi
8. forma `atrybut < wartość`
   na liście sprawdzanych elementów sprawdza, czy wartość atrybutu w zleceniu jest mniejsza od podanej
   NIE MOŻE wystąpić na liście odpowiedzi
9. forma `atrybut <= wartość`
   na liście sprawdzanych elementów sprawdza, czy wartość atrybutu w zleceniu jest mniejsza lub równa podanej
   NIE MOŻE wystąpić na liście odpowiedzi
10. forma `atrybut =~ wyrażenie`
    na liście sprawdzanych elementów sprawdza, czy wartość atrybutu dopasowuje się do wyrażenia (wartość znakowa!)
    NIE MOŻE wystąpić na liście odpowiedzi
11. forma `atrybut !~ wyrażenie`
    na liście sprawdzanych elementów sprawdza, czy wartość atrybutu nie dopasowuje się do wyrażenia (wartość znakowa!)
    NIE MOŻE wystąpić na liście odpowiedzi
12. forma `atrybut =* wartość`
    na liście sprawdzanych elementów sprawdza, czy zlecenie zawiera dany atrybut (wartość jest nieistotna)
    NIE MOŻE wystąpić na liście odpowiedzi
13. forma `atrybut !* wartość`
    na liście sprawdzanych elementów sprawdza, czy w zleceniu nie ma danego atrybutu (wartość jest nieistotna)
    NIE MOŻE wystąpić na liście odpowiedzi

Przykłady: (\ oznacza, że c.d. MUSI być w tej samym wierszu)

```
user1 User-Password == "user1pass"
```

```
user2 Auth-Type := Reject
      Reply-Message = "Zablokwane konto"
DEFAULT Realm == NULL, Auth-Type := Reject
DEFAULT Realm = "a.z", Client-IP-Address == 10.1.1.1, \
      Auth-Type := Reject
DEFAULT Real == "b.z" Hundgroup-Name =="XXX"
      Tunnel-Private-Group-Id := 40
      Tunnel-Medium-Type = 6
      Tunnel-Type = VLAN
DEFAULT Real == "c.z" FreeRadius-Proxied-To == 127.0.0.1, \
            Autz-Type := gosc, ldapgosc-Ldap-Group == "z_PL"
      Tunnel-Private-Group-Id := 40
      Tunnel-Medium-Type = 6
      Tunnel-Type = VLAN
```

Użytkownik `user1` (nazwa realmu pusta) zostanie zaakceptowany, jeśli poda hasło `user1pass`.
Użytkownik `user2` (nazwa realmu pusta) zawsze zostanie odrzucony.
Dowolny użytkownik zostanie odrzucony, jeśli realm wynikający z nazwy sieciowej jest pusty.
Dowolny użytkownik, który pochodzi z realmu "a.z" i próbuje uzyskać dostęp za pośrednictwem klienta o IP 10.1.1.1 zostanie odrzucony,
Dowolny użytkownik, który pochodzi z realmu "b.z" zostanie zaakceptowany, jeśli adres klienta należy do grupy XXX. Użytkownikowi zostanie przypisany VLAN 40.
Dowolny użytkownik, który pochodzi z realmu "c.z" i uwierzytelniania się poprzez EAP-TTLS (atrybut **`FreeRadius-Proxied-To==127.0.0.1`**) podlega autoryzacji za pomocą sekcji opisanej w `radiusd.conf` pod nazwą 'gosc' i zostanie zaakceptowany tylko wówczas, gdy atrybut grupowy pobrany z bazy LDAP ma wartość "z_PL". Przyjęty użytkownik zostanie przypisany do VLAN-u 40.

Zalecenia dotyczące pliku `users`:

1. w pakiecie Access-Accept powinna zostać wysłana prawdziwa nazwa użytkownika, dlatego należy stosować konfigurację:
   ```
   DEFAULT Realm = "d.z", ...
         User-Name = `%{User-Name}`
   ```
   pełna nazwa użytkownika zostanie wpisana w logach, np. tych, które powstają po pomyślnym uwierzytelnieniu użytkownika;
2. wpisy w `users` muszą uwzględniać uniknięcie zapętlenia serwera, np.
   ```
   DEFAULT Realm = "pl",Client-IP-Address == 158.75.1.25, \
       Auth-Type := Reject
   ```
   zapobiega przesłaniu zlecenia dot. realmu pl do serwera krajowego, ponieważ stamtąd nadeszło zlecenie.

Plik `radiusd.conf`

W pliku tym jest realizowane ustalenie ścieżek wiodących, lokalizacji konfiguracji, logów itp. Przypisania
```
user = radius
group = radius
```
sprawiają, że serwer będzie pracował z uprawnieniami użytkownika i grupy radius. Nie zaleca się uruchamiania serwera z prawami root (ale jest to niezbędne w przypadku korzystania przez FreeRADIUS-a z Active Directory za pośrednictwem demona winbindd).

Sekcja `listen` ustala `ipaddr` (adres IP / nazwa / *), port (0 - domyślny), type (auth / acct) – w ten sposób można spowodować, by serwer nasłuchiwał na konkretnym interfejsie.

Sekcja `modules` definiuje moduły używane przez serwer. Postać definicji jest następująca:
```
nazwa [ wystąpienie ] {
 element_konfiguracji = wartość
}
```
Przykładowe moduły, które mogą zostać zdefiniowane: `pap`, `chap`, `pam`, `unix`, `mschap`, `ldap`, `sql`, `realm`, `preprocess`, `files`, `detail`, `exec`, `perl`, `eap`, `attr_rewrite`, `attr_filter`.

Sekcja `instantiate` ustala kolejność ładowania modułów – moduły wymienione w niej są ładowane

przed tymi, które są zdefiniowane w następnych sekcjach: `authorize`, `authenticate`, `preacct`, `accounting`, `session`, `post-auth`, `pre-proxy` i `post-proxy`.

Moduł `ldap` pozwala zdefiniować sposób komunikowania się z bazą LDAP w celu autoryzacji i/lub uwierzytelnienia. Jeśli jest potrzeba zdefiniowania kilku modułów `ldap`, kolejne definicje rozróżniamy nazwami wystąpienia modułu, np.:

```
ldap AD { }
ldap OLStaff { }
```

zdefiniowane nazwy mogą być następnie używane w sekcjach `authorize`, `authenticate`.
Jeśli w pliku `users`, używamy sprawdzenia grupy LDAP, to w przypadku stosowania wielu definicji modułów ldap, w elemencie sprawdzenia należy podać wystąpienie modułu, np.:

```
AD-Ldap-Group == "XXX"
```
albo
```
OLStaff-Ldap-Group == "XXX"
```

Zaleca się korzystanie z bezpiecznych połączeń z serwerami LDAP (`start_tls=yes`) oraz nie używanie anonimowych przeszukiwań.
Jeśli dysponujemy serwerem repliki, to należy zadeklarować jego użycie dla danego poddrzewa poprzez utworzenie drugiego wystąpienia modułu, np.

```
ldap OLStaff1 { tu_deklaracje_serwera_głównego }
ldap OLStaff2 { tu_deklaracje_serwera_repliki }
```

W bloku definicji wystąpienia modułu LDAP można zdefiniować parametry `groupmembership_filter` oraz `groupmembership_attribute`.
Parametr `groupmembership_attribute` określa atrybut LDAP stosowany do sprawdzania, czy użytkownik należy do określonej grupy. Najpopularniejszym rozwiązaniem jest stosowanie atrybutu `radiusGroupName` ze schematu RADIUS-LDAPv3.schema, załączonego w dystrybucji FreeRADIUS. Parametr decyduje o sposobie przeszukiwania bazy. Jeśli nie określono `groupmembership_filter`, to stosowany jest domyślny filtr

```
(|(&(objectclass=groupOfNames)(member=%{Ldap-UserDn}))
      (&(objectclass=groupOfUniqueNames)
      (member=%{Ldap-UserDn})))
```

Moduł `ms_chap` jest niezbędny, gdy dostarczamy metodę PEAP (uwierzytelnianie MS-CHAP i MS-CHAPv2). Konfiguracja w sekcji `mschap` powinna być następująca:

```
authtype = MS-CHAP
use-mppe = yes
require_encription = yes
require_strong = yes
```

Moduł `attr_rewrite` bywa stosowany w procesie autoryzacji oraz rozliczania. Umożliwia dokonanie zmian w pakietach.
Wystąpienia modułu są wyróżniane nazwą, np.

```
attr_rewrite copy.user-name { }
attr_rewrite strip.user-name { }
```

Działanie modułu jest następujące:

- zmiany dotyczą określonego atrybutu
  `attribute = ..`
- przeszukiwany jest pakiet, odpowiedź, konfiguracja
  `searchin = packet | reply | config`
- poszukiwana jest wartość zgodna z zadanym wyrażeniem regułowym
  `searchfor = ...`
- dopasowany napis jest zastępowany zawartością określoną jako `replacewith`
- można dopisać nowy atrybut (`new_attribute=yes`)
- można napis zastępujący dopisać do oryginalnego (`append=yes`)
- można zadeklarować maksymalną liczbę dopasowań (`max_matches`)

- można ignorować wielkość liter (ignore_case=yes)

Zdefiniowane wystąpienie modułu dodajemy wg potrzeby, w sekcjach pre-proxy, post-proxy, authorize.
Przykładowe zastosowanie modułu mod_rewrite jest następujące:
Definiujemy wystąpienia copy.user-name i strip.user-name:

```
# obcięcie nazwy przed wyszukaniem użytkownika w
# bazie
attr_rewrite copy.user-name {
attribute =  Stripped-User-Name
new_attribute = yes
searchin = packet
searchfor = ""
replacewith = "%{User-Name}"
}
attr_rewrite strip.user-name {
attribute =  Stripped-User-Name
new_attribute = no
searchin = packet
searchfor = "@.*$"
replacewith = ""
max_matches = 1
}
```

Następnie w module authorize wywołujemy oba moduły:

```
authorize {
..
copy.user-name
strip.user-name
...
}
```

Moduł attr_filter jest przeznaczony do filtrowania atrybutów. Może zostać wywołany m.in. po odbiorze odpowiedzi z serwera proxy, w sekcji post-proxy.
W sekcji attr_filter w pliku radiusd.conf, element konfiguracji attrsfile wskazuje plik z deklaracją działania filtru (domyślnie plik o nazwie attrs). Domyślne działanie oznacza usunięcie z odpowiedzi wszystkich atrybutów oprócz tych, dla których określono działanie w pliku z filtrem.
Możliwa jest poprawka kodu modułu src/modules/rlm_attr_filter/rlm_attr_filter.c (wdrożona na UMK), która pozwala na odwrócenie działania: atrybuty dla których nie określamy obsługi są przekopiowywane.
Przykład:
Na serwerze jednostki włączamy attr_filter w sekcji post_proxy, plik attrs ma postać:

```
pl
 Tunnel-Private-Group-Id := 42
 Tunnel-Type := VLAN
 Tunnel-Medium-Type := 6
 Tunnel-Private-Group-Id !* ANY
 Tunnel-Type !* ANY
 Tunnel-Medium-Type !* ANY
DEFAULT
```

Celem tego filtra jest wycięcie VLAN-ów ustawionych na obcym serwerze i ustawienie VLAN-u 42.

Moduł perl pozwala wskazać skrypt perl, który może służyć np. do filtrowania pakietu, realizacji uwierzytelnienia, autoryzacji itp. Definicja polega na dodaniu sekcji perl o postaci:

```
 perl {
  module = "/opt/FreeRADIUS/perl/vlans.pl"
  func_post_auth = "post_auth"
 }
```

Skrypt "/opt/FreeRADIUS/perl/vlans.pl" realizuje wszystkie potrzebne zadania.
Elementy konfiguracyjne `func_post_auth`, `func_pre_proxy`, `func_post_proxy` itd. podają nazwy funkcji Perl zdefiniowanych dla odpowiednich sekcji.

Sekcja `authorize`
Sekcja ustala sposób autoryzacji użytkowników. Przykładowa postać to:

```
authorize {
 preprocess
 copy.user-name
 strip.user-name
 Autz-Type pracownik {
   ldapstaff
 }
 Autz-Type student {
   ldapstud
 }
 eap
 files
}
```

Zdefiniowano typy autoryzacji (Autz-Type) pracownik i student, powiązane z wystąpieniami modułu ldap. Autoryzacja odbywa się na podstawie typu określonego przez moduł eap oraz na podstawie modułu files.

Sekcja `authenicate`
Sekcja ustala sposób przebiegu uwierzytelnienia. Przykładowa postać to:

```
authenticate {
 eap
 Auth-Type pracownik {
   ldapstaff
 }
 Auth-Type student {
   ldapstud
 }
 Auth-Type MS-CHAP {
  mschap
 }
}
```

Zdefiniowano typy uwierzytelniania (`Auth-Type`) pracownik i student, powiązane z wystąpieniami modułu `ldap` oraz z modułem `mschap`. Uwierzytelnienie odbywa się na podstawie typu określonego przez moduł `eap`.

Sekcja `post-auth`
W sekcji tej domyślnie jest jedynie wywoływany moduł `reply_log` w celu logowania wszystkich pomyślnych prób uwierzytelnienia i autoryzacji. W tej sekcji można dodać wywołanie modułu perl w celu usunięcia ustawionych wcześniej VLAN-ów w przypadku, gdy Access-Accept z naszego serwera jest kierowany poza sieć danego dostawcy. W tym celu niezbędne jest dokonanie modyfikacji źródeł modułu `src/modules/rlm_perl/rlm_perl.c` pozwalającej na wywołanie modułu w sekcji `post_auth`.
W sekcji `post-auth` można dodać wywołanie modułu files by możliwie najpóźniej (w chwili odsyłania odpowiedzi akceptującej) zrealizować przypisanie VLAN-ów, tylko wtedy, gdy zlecenie dołączenia do sieci wiąże się z siecią danego dostawcy. Takie zastosowanie modułu files wymaga również modyfikacji źródeł – tym razem modułu `src/modules/rlm_files/rlm_files.c`.
Plik `users`, używany w fazie `post-auth` powinien mieć wpisy związane z dopasowaniem w fazie obsługi pakietu Access-Accept:

```
DEFAULT Real == "c.z" Autz-Type := gosc
   Fall-Through = yes
DEFAULT Real == "c.z",  Response-Packet-Type == Access-Accept,\
   NAS-IP-Address != 158.75.1.25,   NAS-IP-Address != 150.254.173.30
```

```
      Tunnel-Private-Group-Id := 40
      Tunnel-Medium-Type = 6
      Tunnel-Type = VLAN
```

Plik `eap.conf`

Plik tej jest włączany do konfiguracji w ramach pliku `radiusd.conf`. Zawiera definicję modułu `eap`. Ustala `default_eap_type` (`peap`, `ttls`, `tls`), domyślny typ EAP-a, definiuje wspierane typy EAP:

```
md5 {
}
peap {
 default_eap_type = mschapv2
}
tls {
 # certyfikat CA, certyfikat serwera,
 # klucz prywatny
 # jeśli jest obsługa CRL check_crl = yes
check_cert_cn = %{User-Name}
check_crl = yes
}
ttls {
 default_eap_type = md5
 copy_request_to_tunnel = yes
 use_tunneled_reply = yes
}
```

Parametr `copy_request_to_tunnel = yes` powoduje, że atrybuty nie zdefiniowane w tunelu, a obecne poza tunelem są kopiowane.
Parametr `use_tunneled_reply = yes` – powoduje, że atrybuty wysyłane w odpowiedzi nie wiążą się z użytkownikiem widzianym na zewnątrz tunelu *(outer identity)*, lecz tym, który został określony wewnątrz tunelu *(inner identity)*.

Certyfikat dla serwera FreeRADIUS

W pliku eap.conf wskazujemy certyfikat serwera, klucz prywatny serwera oraz certyfikat urzędu certyfikacyjnego, który poświadczył certyfikat serwera.
W zleceniu certyfikacji przygotowanemu dla serwera FreeRADIUS w miejscu commonName należy podać pełną kwalifikowaną nazwę serwera.
Urząd wystawiający certyfikat MUSI w certyfikacie serwera dodać rozszerzenie TLS Web Server Authentication:

```
openssl ca -out s.crt -in s.csr -extensions xpserver_ext \
      -exfiles xpextensions
```

Plik `xpextensions` ma postać:

```
[xpserver_ext]
extendedKeyUsage = 1.3.6.1.5.5.7.3.1
```

Ustalenie VLAN-ów

W pakiecie Access-Accept mogą zostać wysłane atrybuty służące do ustalenia docelowego numeru VLAN, w którym użytkownik otrzyma adres. Można to zrealizować za pomocą odpowiednio przygotowanego pliku `users`. Przy obsłudze poszczególnych realmów, można dodać ustawienie atrybutów serii `Tunnel-*`.
uwaga: operator `:=` zamienia dotychczasowe ustawienia dla atrybutu.
Schemat RADIUS-LDAPv3 (fragment dystrybucji FreeRADIUS) definiuje klasę `radiusProfile`. W oparciu o ten schemat FreeRADIUS pozwala na dodanie we wpisie użytkownika informacji o VLAN-ie za pomocą atrybutów: `RadiusTunnelType`, `radiusTunnelMediumType`, `radiusTunnelPrivateGroupID`.
Serwer RADIUS NIE MOŻE przekazywać na zewnątrz ustawień VLAN.

Obsługa rozliczania
Przykładowym rozwiązaniem jest wykorzystanie bazy SQL.
Domyślny plik konfiguracyjny to `sql.conf`. Zawiera on deklaracje dotyczące modułu sql. Są to:
deklaracja drivera, serwera, użytkownika i hasła do współpracy z bazą, nazwy bazy oraz definicje: `accounting_start_query`, `accounting_update_query`, `accounting_stop_query` – sposobu realizacji zapytań.
Należy zainicjować bazę danych, utworzyć tablicę np. o nazwie `accounting`.
W tablicy umieszczamy pola, których dotyczą definicje zapytań `accounting_start_query`, `accounting_update_query`, `accounting_stop_query`.
W sekcji `accounting` dopisujemy wywołanie modułu `sql`.
Szczegółowa instrukcja jest dostępna w języku angielskim, w dokumencie
http://www.terena.nl.mail-archives/mobility/pdf00102.pdf

Uruchomienie serwera FreeRADIUS

Jeśli konfiguracja nie jest w domyślnym katalogu `etc/raddb`
```
radiusd -d katalog_konfiguracji
```

Jeśli uruchamiamy domyślnie
```
radiusd &
```

Polecenie:
```
radiusd -X >& /tmp/freeradius&
```
uruchamia serwer w trybie debugowania, przy czym wyjście debugowania jest przekazywane do pliku `/tmp/freeradius.`

Integracja FreeRADIUS-a z Active Directory

Założenia są następujące:

- korzystamy z Active Directory (Windows Server 2000 / 2003) jako bazy użytkowników;
- użytkownicy mają zdefiniowany tryb 'allow access' (zakładka Dial-In we własnościach konta);
- rozszerzamy schemat poprzez dopisanie klasy `radiusProfile` oraz atrybutów dialupAccess, radiusGroupName, radiusTunnel*;
- rozbudowujemy wpis użytkownika w celu dodania nowych atrybutów;

Uwaga: możliwość zastosowania atrybutów klasy radiusProfile przy opisie kont użytkowników ustala własność klasy `radiusProfile` w ramach zakładki Relationship - jako 'possible superior' należy dodać klasę obiektów `user`.
W celu aktualizacji schematu Active Directory muszą być spełnione warunki:

- administrator lub użytkownik modyfikujący musi być w grupie (Member Of) Schema Admins;
- należy wykonać polecenie schmgmt.dll: Run... regsvr32 schmgmt.dll;
- otworzyć konsolę (mmc) i dodać przystawkę "Active Directory Schema";
- aby była możliwość zapisu aktualizacji niezbędna jest zmiana w rejestrze:
  dojść do HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Services\NTDS\Parameters
  Edit, New, DWORD Value,
  Value Name: Schema Update Allowed
  Data Type: REG_DWORD, Base: Binary, Value Data: 1

Przygotowanie Active Directory:
Konta użytkowników danego kontrolera domeny znajdują się w gałęzi `cn=Users`.
Korzeń drzewa to `dc=a,dc=b,dc=c`, jeśli Active Directory jest kontrolerem domeny `a.b.c`
Program `ldifde` pozwala importować i eksportować dane, domyślny tryb to eksport. Przykłady:
```
ldifde -d 'cn=users,dc=ad,dc=umk,dc=pl' -f u.ldif
```
w pliku `u.ldif` zostanie zapisany zrzut poddrzewa `cn=users`
```
ldifde -f user.ldif -i
```
import danych wg pliku `user.ldif` (wsad w postaci standardu LDIF, DN entry, definicja operacji (changetype: add/modify/ delete), atrybuty.

Dokument “FreeRADIUS Tutorial for AD Integration”, Ch. Schwartz

http://homepages.lu/charlesschwartz/radius/freeRadius_AD_tutorial.pdf
szczegółowo opisuje kolejne działania konfiguracyjne.

Definiujemy moduł ldap, np.:

```
ldap AD {
...
identity = DN_użytkownika z prawem_odczytu
password = "hasło_użytkownika"
start_tls = no
access_attr = "dialupAccess"
...
}
```

W sekcji `authorize` umieszczany odpowiedni `Autz-Type`, podobnie w sekcji `authenticate` umieszczamy `Auth-Type` (by wskazać móc wskazać w `users` obsługę dla danego realmu).
W przypadku współpracy z Active Directory stosowanym typem uwierzytelnienia jest PEAP. Po stronie serwera Linux należy zdefiniować sposób dostępu do domeny Windows.
Korzystamy w tym celu z serwera Samba i narzędzi:

- `winbindd`: demon umożliwiający połączenie ze środowiskiem Windows,
- `ntlm_auth`: program korzystający z `winbindd` do realizacji zlecenia NTLM .

W pliku `/etc/krb5.conf` należy dodać domenę Windows

```
AD.X.Y {
kdc = 192.168.3.33
default_domain = AD.X.Y
}
```

Po stronie Samby, w pliku smb.conf umieszczamy w sekcji global:

```
[global]
workgroup = AD.UMK.PL
security = ads
...
password server = 192.168.3.33  // serwer AD
realm = AD.UMK.PL
```

Restartujemy serwer `smbd`.

W celu rejestracja serwera w domenie Windows wykonujemy:

```
net join -U administrator
```

Następnie uruchamiamy demona `winbindd`.

Wykonujemy próbę uwierzytelnienia przez NTLM

```
ntlm_auth -request-nt-key -domain=AD.X.Y -username=xxx
```

Dostosowujemy konfigurację serwera FreeRADIUS:

- w sekcji mschap musi być wskazany `authtype = MS-CHAP`,
- odkomentowujemy definicje `with_ntdomain_hack = yes`
- definiujemy:

```
ntlm_auth = "/usr/bin/ntlm_auth -request-nt-key
      --domain=AD.UMK.PL -username=%{Stripped-User-Name}
      --challenge=%{mschap:Challenge}
      --nt-response=%{mschap:NT-Response}"
```

- w pliku `eap.conf` ustawiamy:

```
default_eap_type = peap
```

w sekcji `tls` konieczna jest deklaracja certyfikatów (PEAP korzysta z zaszyfrowanego tunelu);
należy odkomentować sekcję `peap`

```
peap {
default_eap_type = mschapv2
}
```