Contents

- 1 Xmlcdrd FastCGI CDR Logger
  - 1.1 Overview
  - 1.2 How does it work
  - 1.3 Requirements
  - 1.4 Configuration overview
    - 1.4.1 Mod_xml_cdr Configuration
    - 1.4.2 mysqlcdr plugin
    - 1.4.3 luaexec plugin
    - 1.4.4 radcdr plugin
  - 1.5 Build
  - 1.6 More complex example of radius accounting with stop packets

Xmlcdrd FastCGI CDR Logger

Overview

Xmlcdrd is a FastCGI daemon that works together with mod_xml_cdr to process CDR records. Right now xmlcdrd can store CDR fields into MySQL tables, submit Radius accounting (stop) packets and run custom Lua scripts on CDR. Functionality of xmlcdrd could be extended by writing plugins for other databases or different script languages,so expect ODBC plugin soon.

How does it work

Xmlcdrd is run with FastCGI compliant web server. It has been tested with lighttpd version 1.4, but could work with any server which supports FastCGI. When fcgi receives the CDR, it parses it and stores the configured XML entries (variables) into a hash table. Then it matches variables against "plugin metrics" and passes that hash table to plugins main function on successful match. Serveral instances of xmlcdrd can be started if one instance is not enough for CDR processing. All xmlcdrd logging goes to the stderr, so it's inside web servers error logs.

Requirements

Mandatory:

- FastCGI compliant web server. Lighttpd 1.4 is highly recommended.
- Apache libraries: apreq2 and aprutil1
- Libconfig library
- Fast-CGI library
- Glib2 library
- GNOME libxml2 library

Optional:

- Database Server: Mysql server / mysql C client libs for mysqlcdr plugin. Luaexec can use virtually any database via luasql.odbc and unix-odbc.

Configuration overview

Variables and plugins are configured in a main configuraton file xmlcdr.conf. You can declare as many variables as you want.

Variables are defined using XPATH expressions.

Example Variables section:

```
#Variables definition
Variables = (
 {
  name = "uuid";
  var_id = "uuid";
  xpath = "/cdr/callflow[@profile_index='1']/caller_profile/uuid";
 },
...
 {
  name = "effective_caller_id_number";
  var_id = "effective_caller_id_number";
  xpath = "/cdr/variables/effective_caller_id_number";
 }
);
#End of variables section
```

Plugins are executed depending on plugin metrics and can access CDR variables. Metrics are defined using regexps. Each plugin may have a multiple metrics. All metrics must evaluate to true for successful execution. Plugin with no metrics will be executed everytime data is received. Each plugin might be loaded several times with a different config file. This could be useful when logging to a different tables/schemas.

```
#Plugins section
Plugins = (

#Plugin declaration
{
  #name is a name of so file without the extension
  name="mysqlcdr";

  #config file for plugin
  options = "/etc/xmlcdrd/db1.conf";
  priority = 10;

  #Metrics control plugin execution
  #Plugin with no metrics defined will be executed everytime
  Metrics = (
        {
       metric = "destination_number";
       regexp = "^6969.*";
```

```
        }
  );
  #end of metrics

},
{
  #This plugin is executed on all CDRs because it has no metrics

  name="luaexec";
  #config file for plugin
  options="/etc/xmlcdrd/luaexec.conf";
  priority=20;
}

#End of plugin declaration
#You could define multiple plugins
#Plugin declarations are separated with comma. See libconfig manual for details.

);
```

Mod_xml_cdr Configuration

```
Make sure that encode and disable-100-continue are active in xml_cdr.conf.xml
<param name="encode" value="true"/>
<param name="disable-100-continue" value="true"/>

Enable the fast cgi in lighttpd config:

"xmlcdr.fcgi" =>
    ( "localhost" =>
        (
          "socket"   => "/tmp/lighttpd-cdr-fcgi.socket",
          "max-procs" => 1,
          "bin-path" => "/usr/local/bin/xmlcdrd.fcgi",
          "check-local" => "disable"
        )
    )
```

mysqlcdr plugin

Mysqlcdr plugin fills configured variables into a SQL statement and then executes it. All variables are escaped with mysql_real_escape_string function. Plugin accepts single initialization option - name of a configuration file. Example configuration file:

```
#mysqlcdr example config file
dbuserid="ddp";
dbpasswd="password";
```

```
dbschema="schema";
dbipaddr="localhost";
dbipport=3306;

stmt_template="insert into cdr (caller_id_number, effective_caller_id_number, destination_number,

#end of mysqlcdr config file
```

luaexec plugin

Luaexec plugin allows you to write a custom scripts for CDR processing. Lua has a rich set of addon libraries for accessing file system, databases, e.t.c. You need a Lua shared library to build the plugin. If you don?t like the idea of CDR processing with interpreter, ?configure? the build with ?disable-luaexec parameter. Lua is fast and can be precompiled into a byte-code for faster execution. Right now Luaexec is not very optimized for speed, but I?m able to make around 100 inserts per second with a single instance of xmlcdrd and luasql.odbc addon. Inserts should run even faster on a fine tuned database and descent hardware. Lua script must define following functions: function luaexec_init(), luaexec_main() and luaexec_free(). luaexec_init() and luaexec_free() are called only once on initialization and destruction of plugin. Function luaexec_main() is called every time plugin is executed. Luaexec exports two functions: xcdr_varget("name_of_variable") - returns the value of variable given the name of variable or NULL if variable was not found. xcdr_prn_debug ("debug str") - sends the string to syslog with DEBUG level.

```
#Dumb example of Lua script
require "luasql.odbc"

env = nil
db_conn = nil
count = 0

function luaexec_init ()
  env = assert(luasql.odbc())
  db_conn = assert(env:connect("mysqlconnection") )
  count = 1
end


function luaexec_free ()
  db_conn:close()
  env:close()
end

function luaexec_main ()

  uuid = xcdr_varget("uuid")
  billsec = xcdr_varget("billsec")

  db_conn:execute(string.format("insert into cdr (uuid,billsec) values ('%s', %d)", uuid, billsec
  -- notice that billsec will grow
  count = count + 1
end

#End of example Lua script
```

radcdr plugin

Radcdr plugin is based upon Tihomir Culjagas mod_rad_auth. It creates radius stop packets out of FreeSWITCH cdrs. Radcdr depends on freeradius-client library, you may find it here: http://freeradius.org/freeradius-client/ Below is an example configuration file for radcdr plugin.

```
dictionary="/etc/xmlcdrd/radius/dictionary";
seq_file="/tmp/radcdr_seq";
config_file="/usr/local/etc/radiusclient/radiusclient.conf";


#servers section
servers = (

{
host = "172.16.31.8:1813:password";
}
);
#End of servers


vsas = (
   {
        var_name = "caller_id_number";
        vsa_name = "callerid";
        vsa_id = 1;
        vsa_pec = 0;
        vsa_type = "string";
   },
   {
        var_name = "h323-call-origin";
        vsa_name = "h323-call-origin";
        vsa_id = 26;
        #9 Cisco
        vsa_pec = 9;
        vsa_type = "string";
   },
   {
        var_name = "h323-call-type";
        vsa_name = "h323-call-type";
        vsa_id = 27;
        #9 Cisco
        vsa_pec = 9;
        vsa_type = "string";
   }
);
```

Build

Code is located under freeswitch-contrib Git tree nazim/xmlcdrd.

To pool entire tree use: git clone git://git.freeswitch.org/freeswitch-contrib.git

To build use ./bootstrap.sh; ./configure; make; make install

To disable the compilation of particual plugin use --disable switch when running configure.

```
 --disable-mysqlcdr      Disable building of mysqlcdr plugin
 --disable-luaexec       Disable building of luaexec plugin
 --disable-radcdr        Disable building of radcdr plugin
```

More complex example of radius accounting with stop packets

This is an example of radius accounting with radcdr plugin. Please note that this configuration sends only "stop" radis accounting packets "start" packets are not supported. A lot of billing systems support accounting in that way, so absent "start" packets is not an issue in most cases. Xmlcdrd allows you to include different VSA's in a "stop" packets, depending on a call CDR variables.

Let's assume you have to process calls coming to a three extensions: "national", "international" and "disa".

For A-leg Xmlcdrd must generate a radius "stop" packets with a slightly different VSA set, depending on which extension call was landed. A-leg "stop" packets must not include h323-remote-address (vsa_id = 23 / vsa_pec = 9)

For B-leg stop packets must not include user-name (vsa_id = 1/ vsa_pec = 0) VSA, but h323-remote-address must be set, otherwise your supper-billing solution will not process that "stop". Note, "log-b-leg" must be enabled in xml_cdr.conf, otherwise cdr will not be generated for a B-leg.

First we have to declare a variable "extension_name" in a variables section, this variable will contain the extension name for each incoming call. Note, extension_name is meaningful for A-leg only, for B-leg it will be empty.

```
{
name = "extension_name";
var_id = "extension_name";
xpath = "/cdr/callflow[@profile_index='1']/extension/@name";
}
```

Then we will be able to match the extensions name for any plugin.

```
...
        ,
        {
                metric = "extension_name";
                regexp = "^international$";
```

```
        }
...

Sample xmlcdrd.conf

----

#Variables section
Variables = (
{
name = "direction";
var_id = "direction";
xpath = "/cdr/channel_data/direction";
}
,
{
name = "uuid";
var_id = "uuid";
xpath = "/cdr/callflow[@profile_index='1']/caller_profile/uuid";
}
,
{
name = "context";
var_id = "context";
xpath = "/cdr/callflow[@profile_index='1']/caller_profile/context";
}
,
{
name = "effective_caller_id_number";
var_id = "effective_caller_id_number";
xpath = "/cdr/variables/effective_caller_id_number";
}
,
{
name = "caller_id_number";
var_id = "caller_id_number";
xpath = "/cdr/callflow[@profile_index='1']/caller_profile/caller_id_number";
}
,
{
name = "destination_number";
var_id = "destination_number";
xpath = "/cdr/callflow[@profile_index='1']/caller_profile/destination_number";
}
,
#extension_name is for matching extensions name on an inbound call leg
#It's empty on outbound call leg
{
name = "extension_name";
var_id = "extension_name";
xpath = "/cdr/callflow[@profile_index='1']/extension/@name";
}
,
{
name = "callgroup";
var_id = "callgroup";
xpath = "/cdr/variables/planeta_callgroup";
}
,
{
name = "service_type";
var_id = "service_type";
```

```
xpath = "/cdr/variables/service_type";
}
,
{
name = "startep";
var_id = "startep";
xpath = "/cdr/variables/start_epoch";
}
,
{
name = "answep";
var_id = "answep";
xpath = "/cdr/variables/answer_epoch";
}
,
{
name = "endep";
var_id = "endep";
xpath = "/cdr/variables/end_epoch";
}
,
{
name = "h323-call-type";
var_id = "h323_call_type";
xpath = "/cdr/variables/h323_call_type";
}
,
{
name = "h323-call-origin";
var_id = "h323_call_origin";
xpath = "/cdr/variables/h323_call_origin";
}
,
{
name = "USERNAME";
var_id = "USERNAME";
xpath = "/cdr/variables/USERNAME";
}
,
{
name = "CALLINGNUMBER";
var_id = "CALLINGNUMBER";
xpath = "/cdr/variables/CALLINGNUMBER";
}
,
{
name = "ORIGDEST";
var_id = "ORIGDEST";
xpath = "/cdr/variables/ORIGDEST";
}
,
{
name = "startstamp";
var_id = "startstamp";
xpath = "/cdr/variables/start_stamp";
}
,
{
name = "endstamp";
var_id = "endstamp";
xpath = "/cdr/variables/end_stamp";
}
```

```
,
{
name = "billid";
var_id = "billid";
xpath = "/cdr/variables/billid";
}
,
{
name = "h323-remote-address";
var_id = "h323-remote-address";
xpath = "/cdr/variables/h323-remote-address";
}

);
#End of Variables


#Plugins directory
plugin_dir = "/usr/local/lib/xmlcdrd";



#Plugins section
Plugins = (


{
 #Retail international traffic (SIP acct)
 #match only incomin calls landed on an "international" extension
 #international is a name of the extension

  name="radcdr";

  #config file for plugin
  options = "/etc/xmlcdrd/radcdr.conf";
  priority = 100;
  Metrics = (
        {
                metric = "destination_number";
                regexp = "^00.*";
        }
        ,
        {
                metric = "context";
                regexp = "^sip.mydomain.org$";
        }
        ,
        {
                #match A-leg only
                metric = "direction";
                regexp = "^inbound$";
        }
        ,
        {
                metric = "extension_name";
                regexp = "^international$";
        }
  );

}
,
{
```

```
  #Retail national traffic (SIP)
  #matches numbers beginning with 0
  #extension_name = national

  name="radcdr";

  #config file for plugin
  options = "/etc/xmlcdrd/radcdr.conf";
  priority = 110;
  Metrics = (
        {
                metric = "destination_number";
                regexp = "^0[1-9]\d\d\d\d\d\d";
        }
        ,
        {
                #match A-leg only
                metric = "direction";
                regexp = "^inbound$";
        }
        ,
        {
                metric = "extension_name";
                regexp = "^national$";
        }

  );
}
,
{
  #Retail traffic (DISA script)
  #matches numbers beginning with 0 and extension name "disa"

  name="radcdr";

  #config file for plugin
  options = "/etc/xmlcdrd/radcdrdisa.conf";
  priority = 103;
  Metrics = (
        {
                #ORIGDEST is exported from a dialplan extension
                metric = "ORIGDEST";
                #regexp = "^0[1-9]\d\d\d\d\d\d$|^00.*";
                regexp = "^0.*";
        }
        ,
        {
                #match A-leg only
                metric = "direction";
                regexp = "^inbound$";
        }
        ,
        {
                metric = "extension_name";
                regexp = "^disa$";
        }

  );
}
,
#########
#########      Outbound traffic
```

```
##########
{
  #Upstream1 national traffic, origination
  name="radcdr";

  #config file for plugin
  options = "/etc/xmlcdrd/radcdrupstream1.conf";
  priority = 210;
  Metrics = (
        {
                #match B-leg only
                metric = "direction";
                regexp = "^outbound$";
        }
        ,
        {
                metric = "destination_number";
                regexp = "^001431380.*";
        }
        ,
        {
                metric = "h323-remote-address";
                regexp = "^1.1.1.1$";
        }
  );

}
,
{
  #Upstream2 traffic (origination), operator2
  #7 or more digits, first digit non 0, outbound

  name="radcdr";

  #config file for plugin
  options = "/etc/xmlcdrd/radcdrupstream2.conf";
  priority = 220;
  Metrics = (
        {
                #match B-leg only
                metric = "direction";
                regexp = "^outbound$";
        }
        ,
        {
                metric = "destination_number";
                regexp = "^[1-9]\d\d\d\d\d\d.*";
        }
        ,
        {
                metric = "h323-remote-address";
                regexp = "^2.2.2.2$";
        }
  );

}

);
```

Now lets look at the sample configuration for a B-leg radius "stop". Note that "user-name" (vsa_id = 1/ vsa_pec = 0) is not defined in radcdrupstream2.conf, so it will not be sent. For A-leg configuration files, "user-name" VSA will be defined, but not the "h323-remote-address".

```
/etc/xmlcdrd/radcdrupstream2.conf (Config for a B-leg)

----


#USERNAME metric MUST NOT present in a resulting packet!
#h323-remote-address MUST present in a resulting packet!


dictionary="/etc/xmlcdrd/radius/dictionary";
seq_file="/var/tmp/radcdr_seq";
config_file="/usr/local/etc/radiusclient/radiusclient.conf";


#servers section
servers = (

{
host = "172.6.1.8:1813:aQe3dfl";
}
);
#End of servers


vsas = (
   {
        var_name = "service_type";
        vsa_name = "service_type";
        vsa_id = 6;
        vsa_pec = 0;
        vsa_type = "integer";
   },
   {
        var_name = "h323_call_origin";
        vsa_name = "h323-call-origin";
        vsa_id = 26;
        #9 Cisco
        vsa_pec = 9;
        vsa_type = "string";
   },

   {
        var_name = "h323-remote-address";
        vsa_name = "h323-remote-address";
        vsa_id = 23;
        #9 Cisco
        vsa_pec = 9;
        vsa_type = "string";
   },
   {
        var_name = "h323_call_type";
        vsa_name = "h323-call-type";
        vsa_id = 27;
        #9 Cisco
        vsa_pec = 9;
```

```
        vsa_type = "string";
    },
    {
        var_name = "billid";
        vsa_name = "h323-conf-id";
        vsa_id = 24;
        #9 Cisco
        vsa_pec = 9;
        vsa_type = "string";
    },
  {
        var_name = "h323_connect_time";
        vsa_name = "h323-connect-time";
        vsa_id = 28;
        vsa_pec = 9;
        vsa_type = "string";
    },
   {
        var_name = "h323_diconnect_time";
        vsa_name = "h323-disconnect-time";
        vsa_id = 29;
        vsa_pec = 9;
        vsa_type = "string";
    },
  {
        var_name = "billsec";
        vsa_name = "acct_session_time";
        vsa_id = 46;
        vsa_pec = 0;
        vsa_type = "integer";
    },
   {
        var_name = "destination_number";
        vsa_name = "called_station_id";
        vsa_id = 30;
        vsa_pec = 0;
        vsa_type = "string";
    },
   {
        var_name = "CALLINGNUMBER";
        #vsa_name = "calling_station_id";
        vsa_name = "calling_station_id";
        vsa_id = 31;
        vsa_pec = 0;
        vsa_type = "string";
    }

);
```