

## Contents

- [1 Introduction](#)
- [2 TODO](#)
- [3 Replicating Sofia Registrations](#)
  - ◆ [3.1 Configuration](#)
- [4 Example Events](#)
- [5 Other Custom Events](#)
- [6 Event Encryption](#)
- [7 Troubleshooting](#)
  - ◆ [7.1 "Couldn't register subclass"](#)
  - ◆ [7.2 "Bind Error"](#)
  - ◆ [7.3 "Failed to find start of magic string"](#)
  - ◆ [7.4 "Cannot use pre shared key encryption without OpenSSL support"](#)
- [8 Using multicast in networked FSs](#)
  - ◆ [8.1 Introduction](#)
  - ◆ [8.2 Multicast configuration file](#)
  - ◆ [8.3 Lua configuration file](#)
  - ◆ [8.4 Lua scripts](#)

## Introduction

mod\_event\_multicast sends FreeSWITCH events from the machine via multicast to a configurable address/port combination.

Other hosts can be configured to listen for these events and parse them, potentially also triggering events to happen on those hosts.

You get access to all of the same events that you would get with [mod\\_event\\_socket](#) and "event plain ALL". **This can be both a blessing and a curse.** Please take care to audit the events, as they'll be sent everywhere that the router of your subnet and the adjoining routers are configured to send multicast packets. You may wish to explicitly plan for this and use a [VLAN](#) or similar precaution to protect yourself from information leaking to places it need not go.

## TODO

Some TODOs for this module that I ([Vagabond](#)) am interested in adding:

- Config file reloading (so you can change the events being multicasted, mainly) - **Sorta done**
- Use pre shared key encryption to encrypt the packets for security and to allow 2 groups of FreeSWITCH machines to share the same IP/Port combination without stepping on each other - **Done, I think**
- Use multicasted HEARTBEAT events to determine when peers go up/down and generate a custom internal event to indicate this - **Sorta Done**
- Make the TTL on the packets configurable - **Done**

## Replicating Sofia Registrations

As an alternative to using ODBC and a shared database to replicate sofia registrations, you can also use mod\_event\_multicast. This lets you replicate sofia registrations without the requirement of sharing a database which may not always be available (if, for example, one of your machines is in a different location).

### Configuration

On each machine you want to *\*broadcast\** registrations on, you'll need the following:

- mod\_event\_multicast compiled and loaded
- mod\_sofia compiled and loaded (this is default)
- "CUSTOM sofia::register" in the "bindings" parameter of event\_multicast.conf (you can put other events here too)

On each machine you just want to *\*receive\**, you just need mod\_event\_multicast and mod\_sofia compiled and running.

Now, once you register a phone on one machine, you should be able to see the new registration on any of the other machines listening on that multicast IP/port combo.

### Example Events

Events are the same as on the [Event list](#) page except that all original headers are prefixed with 'Orig-' and the event is of type CUSTOM with a subtype of multicast::event. A Multicast-Sender header is also added. Here's a before-after example:

Before:

```
Event-Subclass: sofia%3A%3Aregister
Event-Name: CUSTOM
Core-UUID: 662db344-5ecc-4eaa-9002-9992b7ab7c4d
FreeSWITCH-Hostname: DEV-CS2
FreeSWITCH-IPv4: 192.168.1.15
FreeSWITCH-IPv6: %3A%3A1
Event-Date-Local: 2009-06-16%2018%3A15%3A46
Event-Date-GMT: Tue,%2016%20Jun%202009%2022%3A15%3A46%20GMT
Event-Date-Timestamp: 1245190546126571
Event-Calling-File: sofia_reg.c
Event-Calling-Function: sofia_reg_handle_register
Event-Calling-Line-Number: 1113
Event-Subclass: sofia%3A%3Aregister
profile-name: internal
from-user: 1000
from-host: 192.168.1.15
presence-hosts: 192.168.1.15
contact: %221000%22%20%3Csip%3A1000%40192.168.1.23%3A5060%3Bfs_nat%3Dyes%3Bfs_path%3Dsip%253A1000
call-id: 002D61B2-5F3A-DD11-BF4B-00132019B750%40192.168.1.23
rpid: unknown
statusd: Registered(UDP-NAT)
expires: 900
to-user: 1000
```



## Mod\_event\_multicast

seconds a multicast::peerdown event will be fired.

## Event Encryption

If you set the 'psk' parameter in the config file, and you had the openssl development headers installed when you ran ./configure, the packets generated by the module will be encrypted using the blowfish cipher in CBC (cipher block chaining) mode. Only other peers with the same pre-shared key set will be able to decrypt those packets. Events received by a peer in plaintext when it's configured for encryption will be **discarded**.

The use of encryption protects you from malicious event injection as well as allows you to share a multicast/port combination without conflict (if you had some valid reason to do so).

## Troubleshooting

### "Couldn't register subclass"

This probably means that mod\_sofia has already been loaded and grabbed the multicast::event subclass registration. Ensure that mod\_event\_multicast appears before mod\_sofia in your modules.conf.xml.

### "Bind Error"

This means that the module failed to bind to the multicast address to receive events from other FS instances. FreeBSD jails restrict you to one one IP, so for example this will fail there.

### "Failed to find start of magic string"

The event failed to decrypt properly or was received in plain-text when a encrypted event was expected.

### "Cannot use pre shared key encryption without OpenSSL support"

Install the development headers for openssl and re-run ./configure or don't try to use a pre-shared key.

## Using multicast in networked FSs

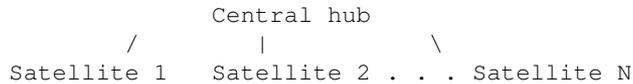
### Introduction

The example below illustrates how the multicast can be used in a networked setup. The scripting language examples are written in Lua.

The topology considered is:

Other Custom Events

## Mod\_event\_multicast



SIP phones are connected to the satellites as well as to the central hub.

Regarding BLF, we want the lamps on a ?local? phone to switch on and off according to other phones busy state ? and this should work even when the other phones are connected to a non-local FS.

Regarding registration, we want the system to route via the hub, if the user is connected to a non-local FS. This is subject to your own design considerations. The dialplan implementing the specific routing is not included in the example.

The event multicast module is used to propagate relevant information among the networked FSs, and Lua scripts listen to the events and act. The network between the switches must support multicasting.

## Multicast configuration file

The configuration file for event\_multicast:

```
<configuration name="event_multicast.conf" description="Multicast Event">
  <settings>
    <param name="address" value="225.1.1.1"/>
    <param name="port" value="4242"/>
    <param name="bindings" value="PRESENCE_IN CUSTOM sofia::register CUSTOM multicast::event"/>
    <!--<param name="ttl" value="1"/>-->
  </settings>
</configuration>
```

It includes three event types: PRESENCE\_IN, which is used to propagate the BLF information; CUSTOM sofia::register used for propagation of registrations; and CUSTOM multicast::event, used for the reception of the two types mentioned.

## Lua configuration file

The events are treated in scripts, in this case Lua scripts, which are started when the FS starts:

```
<configuration name="lua.conf" description="LUA Configuration">
  <settings>
    <param name="startup-script" value="startup_script_1.lua"/>
    <param name="startup-script" value="locationlocal.lua"/>
    <param name="startup-script" value="locationforeign.lua"/>
  </settings>
</configuration>
```

All three scripts listen to and act on events:

startup\_script\_1.lua listens the PRESENCE\_IN events and turns BLF on the local FS on/off accordingly. The locationlocal.lua and locationforeign.lua scripts listens to register events and records in a table (db\_data table in call\_limit core database) the host ip of the FS registered to. This table is then used to lookup the host ip

## Mod\_event\_multicast

during the preparation of the dialstring for local calls.

## Lua scripts

The three scripts are shown below (not thoroughly tested):

```
-- file: startup_script_1.lua
-- Keeps track of foreign presence events and turn on/off
-- busy lamps. Tested on Snom 320. Can easily be
-- changed to include ringing state as well.
con = freeswitch.EventConsumer("CUSTOM","multicast::event")
for e in (function() return con:pop(1) end) do
  -- freeswitch.consoleLog("notice","event\n" .. e:serialize("xml"))
  element = nil
  element = e:getHeader("Orig-status")
  if element then
    if ((element == "CS_EXECUTE") or (element == "CS_ROUTING") or (element == "CS_HANGUP")) then
      event = freeswitch.Event("PRESENCE_IN")
      event:addHeader("proto", "sip")
      event:addHeader("event_type", "presence")
      event:addHeader("alt_event_type", "dialog")
      event:addHeader("Presence-Call-Direction", "outbound")
      from = e:getHeader("Orig-from")
      event:addHeader("from", from)
      event:addHeader("login", from)
      if (element == "CS_HANGUP") then event:addHeader("answer-state", "terminated")
      else event:addHeader("answer-state", "confirmed") end
      event:fire()
      -- freeswitch.consoleLog("notice","event\n" .. e:serialize("xml"))
    end
  end
end

-- file: locationforeign.lua
-- Keeps track of foreign registrations and records them
-- in the db_data table.
conforeign = freeswitch.EventConsumer("CUSTOM","multicast::event")
for e in (function() return conforeign:pop(1) end) do
  -- freeswitch.consoleLog("notice","event:" .. e:serialize("xml") .. "\n")
  element = nil
  element = e:getHeader("Orig-Event-Subclass")
  if element then
    if (element == "sofia::register") then
      registerstatus = e:getHeader("Orig-status")
      if (registerstatus == "Registered(UDP)") then
        ip = e:getHeader("Orig-FreeSWITCH-IPv4")
        name = e:getHeader("Orig-FreeSWITCH-Hostname")
        unitnumber = e:getHeader("Orig-from-user")
        customerid = e:getHeader("Orig-from-host")
        -- freeswitch.consoleLog("notice","Foreign registration: " .. registerstatus .. " ip: " ..
        varfamily = "LOCATION"
        require "luasql.sqlite3"
        env = luasql.sqlite3()
        con = env:connect("/usr/local/freeswitch/db/call_limit.db")
        deletesql = "DELETE FROM db_data WHERE hostname = '..name..' AND realm = '..customerid'"
        result = con:execute(deletesql)
        sql="INSERT INTO db_data (hostname, realm, data_key, data) VALUES ('..name..','..customerid"

```

## Mod\_event\_multicast

```
        result = con:execute(sql)
        -- if (result == nil) then result = '' end
        -- freeswitch.consoleLog("notice","\n" .. "SQL statement: " .. sql .."\n result: " .. result)
        con:close()
        env:close()
    end
end
end
end

-- file: locationlocal.lua
-- Keeps track of local registrations and records them
-- in the db_data table.
conlocal = freeswitch.EventConsumer("CUSTOM","sofia:register")

for e in (function() return conlocal:pop(1) end) do
    -- freeswitch.consoleLog("notice","event:" .. e:serialize("xml") .. "\n")
    registerstatus = e:getHeader("status")
    if (registerstatus == "Registered(UDP)") then
        ip = e:getHeader("FreeSWITCH-IPv4")
        name = e:getHeader("FreeSWITCH-Hostname")
        unitnumber = e:getHeader("from-user")
        customerid = e:getHeader("from-host")
        -- freeswitch.consoleLog("notice","Local registration: " .. registerstatus .. " ip: " .. ip .. "\n")
        varfamily = "LOCATION"
        require "luasql.sqlite3"
        env = luasql.sqlite3()
        con = env:connect("/usr/local/freeswitch/db/call_limit.db")
        deletesql = "DELETE FROM db_data WHERE hostname = '"..name.."' AND realm = '"..customerid..'"
        result = con:execute(deletesql)
        sql="INSERT INTO db_data (hostname,realm,data_key,data) VALUES ('"..name..','".."customerid.."'..unitnumber..','".."customerid.."'..unitnumber..','".."customerid.."'..unitnumber..")
        result = con:execute(sql)
        -- freeswitch.consoleLog("notice","\n" .. "SQL statement: " .. sql .."\n result: " .. result)
        con:close()
        env:close()
    end
end
end
```

In the `modules.xml.conf`, `mod_lua` is the last module loaded. If other modules load after Lua has started, the system may crash.