Call Resolver User Guide

*** DRAFT 4/12/06 ***

Table of Contents

Introduction

The Call Resolver is a sipX component that generates Call Detail Records (CDRs) based on Call State Events (CSEs) logged by a sipX PBX. See the Call Resolver design spec for business/functional requirements and design details. This memo explains how to use the Call Resolver and the outputs that it produces. We also describe how to configure the proxies to log CSEs.

Call Detail Records (CDRs)

Call Resolver output is a set of CDRs stored as records in a PostgreSQL database instance, named "SIPXCDR" by default. It's up to the user to decide how they want to access the data; there are many products out there for database access and data export. For basic database maintenance and inspection, we recommend the "psql" command-line tool included with PostgreSQL, and the graphical tool "pgAdmin III". pgAdmin III is an open source product available from http://www.pgadmin.org/.

As detailed in the design spec, cdrs are stored in two tables, `parties` and `cdrs`. `parties` holds the SIP address of record (AOR) and contact address for callers and callees. An AOR looks like this: "sip:502@example.com", and a contact looks like this: "sip:502@10.1.1.1". "502" is the SIP username, which is typically a phone number or extension, but can also be a name like "alice". Contacts are usually resolved to IP addresses like "10.1.1.1" in this example, but not always.

We provide a view, `view_cdrs`, that presents a simple, unified view of the commonly used CDR data. **Using `view_cdrs` is recommended to insulate you from future database schema changes, as well as simplifying data access.** Here is a simple psql query to print out all the CDRs:

```
psql -d SIPXCDR -U postgres -c "select * from view_cdrs;"
```

Here is an example query on test data, selecting a limited set of columns to fit on the page:

```
psql -d SIPXCDR -U postgres -c "select caller_aor, callee_aor, start_time from view_cdrs;"
```

prints out:

```
      caller_aor        |        callee_aor          |     start_time
------------------------+----------------------------+---------------------
 sip:alice@example.com  | sip:bob@example.com        | 1990-05-17 19:30:00
 sip:mick@example.com   | sip:keith@example.com      | 2001-01-01 00:00:00
 sip:cathy@example.com  | sip:heathcliff@example.com | 1999-08-13 06:00:00
 sip:mick@example.com   | sip:keith@example.com      | 2000-01-01 00:00:00
 sip:alice@example.com  | sip:bob@example.com        | 1990-05-17 19:30:00
(5 rows)
```

Call Direction

"Call direction" is an application-level plugin feature where calls are identified as:

- Incoming (I): the call is coming from a PSTN gateway
- Outgoing (O): the call is going to a PSTN gateway
- Intranetwork (A): neither the caller nor callee is a PSTN gateway

The Call Resolver looks at the From and To contact headers and compares these addresses with the addresses of all gateways configured in sipXconfig, the configuration server. It resolves domain names to IP addresses to get addresses in a standard form where they can be easily compared.

Call direction is stored in the `call_direction` column of the `cdrs` table. There is a view, `view_cdrs_with_call_direction`, that is just like `view_cdrs` but has `call_direction` as an additional column. **Again, accessing the view rather the raw tables protects you from schema changes.** Call direction is not computed by default since most PBXs won't use this feature. See the Configuration section for instructions on how to turn it on.

Call State Events

Call Resolver processes call state events (CSEs) logged by the sipX proxies. CSEs are stored in the same database instance as CDRs. CSE logging must be turned on so that CSEs will be recorded for Call Resolver to operate on. See the section "Configuration" below for details.

Scheduling the Call Resolver

Users do not typically invoke Call Resolver directly, rather it runs automatically every day, by default at 2:47 AM. Each run covers a 24-hour period which (by default) ends at midnight, starting from midnight the previous day. The run time and the time window boundary are both configurable, but the time window duration is always 24 hours.

Not implemented yet, see [XPR-108](XPR-108), will be documented more later

Running the Call Resolver Manually

In a UNIX terminal window:

```
sipxcallresolver.sh --start "2005-12-1T02:47" --end "2005-12-2T02:47"
```

This example runs the Call Resolver on CSEs collected between 2:47 AM on December 1, 2005 and 2:47 AM on December 2, 2005, recording CDRs. `--start` provides the time at which analysis begins and `--end` the time at which analysis ends. `--end` is optional and defaults to 24 hours after the start time. The time format is [ISO 8601](ISO 8601), the same format used by sipX log files.

Purging Old Data

By default, all CSEs and CDRs older than 35 days are discarded, to keep the disk space consumption under control. This time span is configurable. See the "Configuration" section below.

Installation

The Call Resolver must be installed on the same machine as the proxies (forking proxy and auth proxy) and the configuration server. HA configurations with multiple proxies are not yet supported. Future releases will support HA and offer greater flexibility.

Red Hat Enterprise Linux 4 (RHEL4)

Pingtel's commercial SIPxchange™product bundles all dependencies. There are no special installation instructions. Notes:

- The initial SIPxchange™ 3.3 release featuring the Call Resolver does not support upgrading from earlier versions: it must be installed on a clean system.
- Applications on RHEL typically rely on packages installed via RHEL's up2date mechanism. However, the Call Resolver needs a more recent version of the Ruby language interpreter. RHEL provides Ruby 1.8.1; SIPxchange™ upgrades that to Ruby 1.8.4.

Other Linux Distributions

Detailed instructions to be provided later. Here is a start:

- Install **PostgreSQL 7.4** or greater. This requirement is shared with the configuration server, sipXconfig, which cohabits with Call Resolver.
- Install **RubyLibs 1.8.4** or greater. For example, as root on Fedora Core 4: `yum install ruby-libs`.
- Install **Ruby 1.8.4** or greater: `yum install ruby`.
- Install **Ruby Devel 1.8.4** or greater: `yum install ruby-devel`.
- Install **Irb 1.8.4** or greater: 1.8.4 or greater: `yum install irb`.
- Install **RubyGems 0.8.11** or later, from http://rubyforge.org/projects/rubygems/. RubyGems is not generally available in RPM form, although we may make a RubyGems RPM available on sipfoundry for Fedora Core 4.
- RubyGems is a platform-independent packaging mechanism like `yum`. As root, use RubyGems to install remaining dependencies via the Internet:
  - `gem install rails`
  - `gem install postrgres-pr`
- The minimum required versions are **rails 1.0.0** and **postgres-pr 0.4**.
  - Rails is a web application framework. We are primarily using just the ActiveRecord part of Rails, for database access. But when "gem install rails" asks you about installing dependencies, say yes to all dependencies. It's hard to tease out which pieces are needed and which aren't, and not worth the effort given that these packages are small.
  - Postgres-pr is the Ruby database driver for PostgreSQL.
- See the section "System Requirements" for a summary and for info on checking that the right dependencies are installed.
- You don't need to install any of this stuff to get basic PBX functionality.

- **Run sipxcallresolver.sh to set up the SIPXCDR database:**

```
cd /usr/bin
./sipxcallresolver.sh --setup
```

This is handled automatically on RHEL4 when using the install script. Running it again does no harm and can be a useful diagnostic.

Configuration

Call State Event Logging

Call state event (CSE) logging to the database must be turned on for the authorization proxy and forking proxy in order to provide the raw data from which CDRs are created:

- In authproxy-config.in, set logging parameters as follows:
  ```
  SIP_AUTHPROXY_CALL_STATE :
  SIP_AUTHPROXY_CALL_STATE_LOG :
  SIP_AUTHPROXY_CALL_STATE_DB : ENABLE
  ```
- The first two parameters control logging to XML files and should be left blank. Setting `SIP_AUTHPROXY_CALL_STATE_DB` to `ENABLE` turns on call state event logging to the database.
- Similarly, in proxy-config.in, set logging parameters as follows:
  ```
  SIP_PROXY_CALL_STATE :
  SIP_PROXY_CALL_STATE_LOG :
  SIP_PROXY_CALL_STATE_DB : ENABLE
  ```

Call Resolver

All Call Resolver configuration parameters are set in `callresolver-config.in`, located in `/etc/sipxpbx` with the other sipX configuration files.

- **Daily run**: By default, the Call Resolver does not run automatically. To run the Call Resolver every day at 4 AM local time, add this line to the config file:

  ```
  SIP_CALLRESOLVER_DAILY_RUN : ENABLE
  ```

- By default, the Call Resolver analyzes the time window from 4 AM the previous day up to 4 AM on the current day, that is, up to the time that the resolver starts running. To change that time to 3 AM, for example, do this:

  ```
  SIP_CALLRESOLVER_DAILY_START_TIME : 3:00
  ```

  If you pick a time after 4 AM, then the time window will end at that time on the previous day, to avoid losing any data. For example, suppose the start time is 6 AM and the resolver runs on Tuesday night. It will analyze from 6 AM Sunday night to 6 AM Monday night.

- **Call direction:** To configure the Call Resolver to compute call direction, add this line to the config file:

  ```
  SIP_CALLRESOLVER_CALL_DIRECTION : ENABLE
  ```

- **Data purging**: Purging is on by default. To turn it off:

  ```
  SIP_CALLRESOLVER_PURGE : DISABLE
  ```

  - The purge interval defaults to 35 days. Data older than that is deleted as part of the daily run, if purging is enabled. To change it to (for example) 10 days:

    ```
    SIP_CALLRESOLVER_PURGE_AGE : 10
    ```

- **Logging**: for troubleshooting purposes.
  - By default, Call Resolver logs to a file `sipcallresolver.log` in the directory `/var/log/sipxpbx/`. The default log level is NOTICE, with very few log messages. Call Resolver doesn't follow the standard sipX log format because Call Resolver is not real-time and event-driven, like the proxies, so it has different requirements.
  - **Log level**: control the log level via the config parameter `SIP_CALLRESOLVER_LOG_LEVEL`. For example, set the log level to DEBUG if you are having problems and want the maximum amount of information on what's happening.
  - **Log directory**: set `SIP_CALLRESOLVER_LOG_DIR` if you want the log file to go in a directory other than the default.
  - **Log to the console**: set `SIP_CALLRESOLVER_LOG_CONSOLE` to ENABLE to cause logging to go to the console. Unlike other sipX components, logging can go to a file or to the console, but not both.

Troubleshooting

- Do you have all the right software installed? See "System Requirements".
- Is call state event logging turned on? See "Configuration". Look at the table `call_state_events` in the SIPXCDR database and see if there's any data there.
- Try running the Call Resolver script and see if it complains, in this example everything is fine:

  ```
  $ /usr/bin/sipxcallresolver.sh --setup
  Database SIPXCDR exists
  ```

- Call Resolver is installed with unit tests that you can run as a sanity check:

  ```
  $ cd /usr/bin/sipxcallresolver/test/functional/
  $ ruby call_resolver_test.rb

  Loaded suite call_resolver_test
  Started
  ........
  Finished in 0.727583 seconds.

  18 tests, 93 assertions, 0 failures, 0 errors
  ```

  The important thing here is "0 failures, 0 errors" as highlighted above. If the test reports errors, then that information can be useful in diagnosing the problem.

System Requirements

- Red Hat Enterprise Linux 4 or Fedora Core 4.

- Other Linux distributions should work but have not been tested.
- The Call Resolver is 100% portable Ruby code that should run just about anywhere, including Windows and MacOS. But we haven't tested that, and the proxies that generate call state events run only on Linux.
- PostgreSQL 7.4 or greater
- Ruby Libs 1.8.4
- Ruby 1.8.4
- Ruby Devel 1.8.4
- Irb 1.8.4
- RubyGems 0.8.11
- Rails 1.0.0 and its dependencies
- Postgres-pr 0.4

To check the Ruby installation, on Fedora Core 4:

```
$ yum list installed | grep ruby
ruby.i386                          1.8.4-1.fc4          installed
ruby-devel.i386                    1.8.4-1.fc4          installed
ruby-libs.i386                     1.8.4-1.fc4          installed
$ yum list installed | grep irb
irb.i386                           1.8.4-1.fc4          installed
```

Try running Ruby:

```
$ ruby --version
ruby 1.8.4 (2005-12-24) [i386-linux]
```

Use RubyGems to show which “gems” are installed. Most of these gems are Rails and its dependencies:

```
$ gem list

*** LOCAL GEMS ***
actionmailer (1.1.5)
    Service layer for easy email delivery and testing.
actionpack (1.11.2)
    Web-flow and rendering framework putting the VC in MVC.
actionwebservice (1.0.0)
    Web service support for Action Pack.
activerecord (1.13.2)
    Implements the ActiveRecord pattern for ORM.
activesupport (1.2.5)
    Support and utility classes used by the Rails framework.
postgres-pr (0.4.0)
    A pure Ruby interface to the PostgreSQL (>= 7.4) database
rails (1.0.0)
    Web-application framework with template engine, control-flow layer,
    and ORM.
rake (0.7.0)
    Ruby based make-like utility.
sources (0.0.1)
    This package provides download sources for remote gem installation
```