Sip Anomaly Detection Engine (SipADE)

Gurvinder.Singh@uninett.no

May 26 2010, Trondheim

In recent years the Voice over Internet Protocol (VoIP) has gain a lot of attention, due to its benefit to make cheaper voice calls compared to the traditional PSTN network. VoIP mainly uses the Session Initiation Protocol(SIP) with RTP to provide the required functionality. It makes various services e.g. Voice conferencing, IVR etc easily accessible compared to the PSTN network.

Similar to any other system on the connected world, VoIP systems are also susceptible to the attacks by remote/local attacker. The attacker can abuse the system by launching an attack on the system, generally to make expensive calls, which results in a huge sum of bill to pay in a short span of the attacked period. Such abnormal calls can have either high frequency to the expensive destination e.g. International or Toll numbers, or a few number of calls for longer duration where user has to pay large amount of money per second. Sip Anomaly Detection Engine (SipADE) is a detection engine, which detects the anomaly behavior in the outgoing calls pattern. Thus SipADE detects such abnormal calls which can have either high frequency to expensive destination or few numbers of calls, which lasts for longer duration to the expensive numbers.

To detect the anomaly in the call pattern, SipADE uses a behavioral algorithm. The algorithm learns the behavior of organization's call pattern. It trains the algorithm for the specified training period and drives a threshold value for the organization's call traffic. After training phase, SipADE enters in to the detection phase, where it uses the derived threshold value to detect the anomalies. It keep learning the call pattern even in the detection phase, but only from the behavior of the normal call pattern. The learning in detection phase, helps the threshold value to reflect the legitimate changes in the call pattern and avoid false positives.

1. SipADE Architecture

SipADE consists of 5 different module to perform its functionality. The different modules are shown below in the Figure 1.

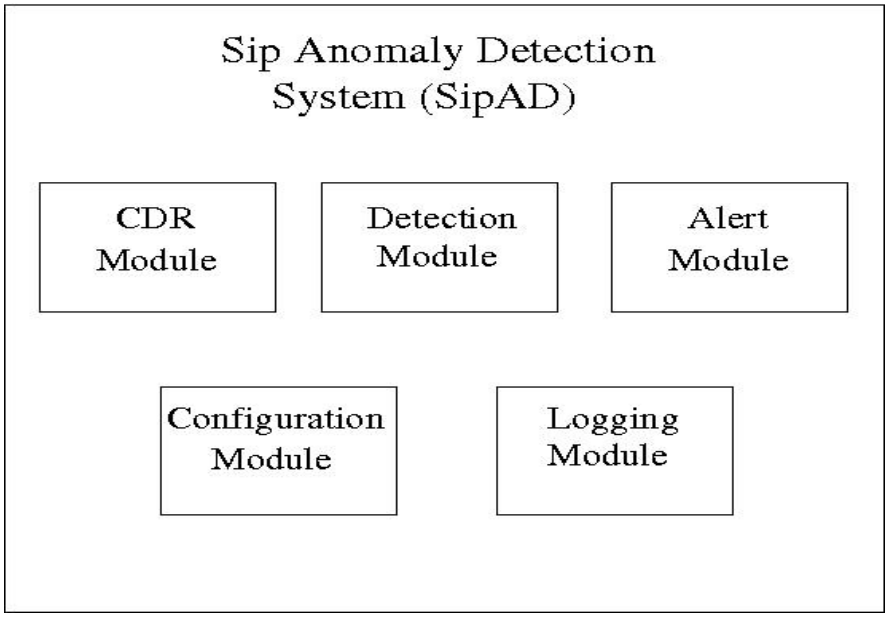

Figure 1: SipADE Architecture

1.1 CDR Module

The CDR module is responsible for making the connection to the CDR database and fetch data from it by making a given query. It fetches the CDR records related to all the monitored call types, which are placed in time interval from the last fetched records plus the specified interval time. The CDR records are logged by Asterisk or any other similar kind of platform. It contains the detail record of the placed call. SipADE uses these call records to detect the anomaly in the call pattern of the organization.

The CDR module needs the following fields with the exact names and type to exist in the CDR database schema to operate correctly. The CDR database can have other fields too, but the fields mentioned below are mandatory. The order of fields mentioned below can be random.

- id serial,
- calldate timestamp with time zone DEFAULT now() NOT NULL,
- src text DEFAULT ''::text NOT NULL,
- dst text DEFAULT ''::text NOT NULL,
- billsec integer DEFAULT 0 NOT NULL,
- accountcode text DEFAULT ''::text NOT NULL,
- calltype text not null default ''::text

where *id* represents the unique id for a CDR record, *calldate* tells us the time and date when the given call was placed, *src* tells us who is the caller, *dst* tells us the destination to which the call is placed, *billsec* tells us the duration of the call which will be billed by the telephony service provider, *accountcode* tells us the institution from where the call has been originated and lastly *calltype* tells us of what type the destination number is. Different types of calls are:

- INTERNATIONAL – calls which are made to numbers out of the country.
- MOBILE – calls which are made to the mobile number inside the country.
- PREMIUM – calls which are made to the toll services number which costs large amount of money. They are the calls to the national toll services numbers.
- SERVICE – calls which are made to the toll services number which does not cost money. They are the calls to the national toll services numbers.
- DOMESTIC – calls which are made to the fixed line national numbers.
- EMERGENCY – calls made to the emergency services such as fire brigade, police or ambulance.

Note that the call type names mentioned above are case sensitive, so while classify and marking the calls to their corresponding types, check the call type name to be of correct case.

1.2 Detection Module

After fetching the data from the CDR database, the CDR data is sent to the detection module. This module uses the behavioral based algorithm to detect anomaly in the fetched data. The algorithm used in the SipADE to detect the anomalous behavior is "*Hellinger Distance*" algorithm[1]. In this algorithm

we calculate the distance between the chosen parameters, which are the different call types. The distance value is a measure of the variability between the two probability distributions of the parameters. The value is calculated between the *probabilities* of the occurrences of desired event during *training period* and during the *testing period*.

$$HD = (sqrt(p) - sqrt(q))^2$$

where p => probability of the desired event during the training period
q=> probability of the desired event during the testing period

p,q is calculated as *number of desired event occurrences* divided by *total number of events* in the given time period. The section 4.1 in [1] gives good overview of using "Hellinger Distance" algorithm for anomaly detection. To use this algorithm in SipADE engine, *different call types* are chosen as the desired event of occurrences. The distance value is calculated for both frequency of the calls and their duration in the given interval. This helps in detecting the attack related to high frequency calls to expensive destination e.g. International calls, whereas the duration values helps in detecting the attack which are related to few calls with large duration to expensive destinations, e.g Toll fraud numbers.

From the calculated distance value, we use the stochastic gradient algorithm to compute the dynamic threshold value. The threshold value is based upon the calculated distance value and derived using the the Jacobson's Fast algorithm for RTT mean and variation calculation [2].

$$Err = m_n - a_{n-1}$$
$$a_n = a_{n-1} + g * Err$$
$$v_n = v_{n-1} + h.\ (|Err| - v_{n-1})$$

Err is the difference between measured HD value (m_n) from this interval and expected HD value (a_n), v_{n-1} and v_n represent the previous and current mean deviations. g and h are the coefficient values to make the expected value of HD with the help of mean deviation to converge as close as possible to the actual HD value. The default values for $g = 1/2^3$ and $h = 1/2^4$. The threshold value is then calculated using the HD value (a_n) and mean deviation (v_n) as shown below:

$$\{Hd_{thres}\}^{n+1} = sensitivity * a_n + adaptability * v_n$$

sensitivity and *adaptability* value tells the algorithm, how much sensitive the threshold value should be to the changes and how much adaptable threshold should be to the call pattern changes. The value of *sensitivity* and *adaptability* should be specified in the configuration file according to the organization call traffic pattern.

The threshold value calculated after the training period is stored in the threshold database. The value is stored after each interval, so that in the case of system crash or engine crash, the threshold value can be stored back quickly. This avoids the unnecessary training on the start of the engine after crash. The threshold database schema to store the threshold value and other related fields is given in the Appendix A.

1.3 Alert Module

SipADE supports two methods to send the alerts after the anomalous behavior has been detected. It can send alerts to the *syslog* or to the *hobbit* [3] server or to *both*. The alert message to be sent to syslog or to hobbit server contains the *time stamp, status message, account code* and the *unique alert ID*. The example alert message will look like this

[2010-01-17 16:37:56] FATAL 59713 1

if there has not been any alert in the given interval then only in the *hobbit* alert mode, it will log the OK status as

[2010-01-17 16:37:56] OK 59713

This helps the hobbit server to keep the service status updated. The information in status message uniquely identifies each alert and the institution for which the alert has been raised. Based on the status message, alert can be sent to the responsible person/administrator at the corresponding institution identified by the account code in the alert message. In the hobbit server, it is easy to configure the method in which alert will be delivered to the responsible person at the institution e.g. via SMS or email etc.

SipADE logs all the CDR records from which anomaly has been detected in the given interval. The records are logged in to the cdr alert database, the information to be stored in the alert database contains following fields.

- alert_id integer DEFAULT 0 NOT NULL,
- cdr_id integer DEFAULT 0 NOT NULL,
- calldate timestamp with time zone DEFAULT now() NOT NULL,
- src text DEFAULT ' '::text NOT NULL,
- dst text DEFAULT ' '::text NOT NULL,
- billsec integer DEFAULT 0 NOT NULL,
- calltype text DEFAULT ' '::text NOT NULL,
- accountcode text DEFAULT ' '::text NOT NULL

The field names are self explanatory and helps the administrator to find out the responsible number/numbers for the alert and take the action accordingly.

1.4 Logging Module

The logging module logs the informative or error messages to the stdout or stderr correspondingly. In the configuration file, the logging mode can specified and depending upon the mode, message belonging to specified mode will be output, except the error message which will always be output to the stderr, irrespective of the specified logging mode. The different logging modes supported by the SipADE are:

- info

- debug
- error

The names of logging mode are self explanatory in themselves. The log messages will contain the time stamp when the message has been output. The info messages will look like this

[21/5/2010 -- 11:01:28] <INFO> Training the engine for detection of anomalous behavior...

the debug messages will look like this

[21/5/2010 -- 11:01:38] <DEBUG> [util-detection.c:260] Duration of Total Calls: 0

The debug message contains the filename, line number from which the message has been printed. Lastly the error messages will look like this

[21/5/2010 -- 11:01:43] <ERROR> [util-detection.c:594] Failed in connecting to threshold-database

The error messages will also contain the filename and the line number, where the error has been occurred.

1.5 Configuration Module

The configuration module is used to parse the parameter's values defined in the configuration file. SipADE uses libyaml to parse the configuration file and get access to the parameters value. The configuration file contains various parameters, which provide a great deal of flexibility in customizing the SipADE to the target organization's requirements. Please make sure while specifying values for the configuration parameters, the values are case sensitive. The following parameters are defined in the configuration file:

Institution name for which we are running the anomaly detection engine.
institution: 59713

The run mode for SipADE engine, the options are "online or offline". In online mode SipADE will run continuously, until it will be killed or system has been crashed. In the offline mode, it will read the given database for the CDR records and will detect the attacks, until the provided ending-date has been reached and after that SipADE will shut down itself.
run-mode: offline
ending-date: '2010-04-01 00:00:00'

The information to make a connection to the CDR. The engine will fetch the cdr records and run the anomaly detection algorithm on them.
cdr-database:
host: localhost
username: asterisk
password: 123456
database-name: asterisk
table: cdr
port: 5432

Similar to the CDR database, the connection information for the Alert and Threshold database should be specified in the configuration file.

The logging level for the SipADE detection engine. The options are
"info, debug or error".

logging-mode: info

The alert mode to be used by the SipADE engine, while sending the alert. The options are "syslog, hobbit or both". In case of *hobbit/both* define the alert file full path and name as provided in the hobbit server configuration.
alert-mode: hobbit
alert-file: /var/log/sip_alert.log

The *call-type* for which you want to run the detection engine. The options are "International, Mobile, Premium, Service,Domestic,Emergency". If you want to run the SipADE engine for all call types, then specify call-type as "All".
call-type: "International,Mobile,Premium"

The *ad-algo* parameter contains various parameters which are specific to the anomaly detection algorithm. The *sensitivity* value tells the engine how much sensitive the engine should be to the changes in the call pattern. Its value should be higher than 1.0. As a general rule, if you want engine to be highly sensitive to the call pattern changes, then sensitivity should be close 1.0. Higher the value of sensitivity is lower the sensitivity is to the call pattern changes. If there is considerable percentage of false positive in the alerts raised by SipADE, then the solution would be to increase the sensitivity value from its previous value.

The *adaptability* value tells the engine how to adapt to the small changes in the call traffic behavior of the monitoring institution. Its value should be in the range of (0.0-1.0). The adaptability value close to 0.0 means, the system is not much adaptive to the changes in the traffic and where the value near to 1.0 means system is highly adaptive. As a general rule, the adaptability value should be somewhere in the range of 0.2-0.8 depending upon the call traffic pattern.

The *interval* value (in minutes) is the time span after which engine scan the placed calls in the duration from last scanned time plus the specified interval. SipADE engine will fetch the CDR records during the specified time span and will run the anomaly detection algorithm on the fetched records.

The *threshold-restore* field tells the engine that should it try to restore the threshold value from the threshold database or not. This will help SipADE in quick restart after crash or system reboot. SipADE stores the threshold value after it completes the training for the first time and keep storing the updated threshold value after each interval. If this option is set to yes, then upon restart the engine will fetch the last threshold value and will enter in to detection phase directly without any need of training the detection algorithm.

The *call-freq* field tells the engine about minimum number of calls which are allowed for the organization in a time span as specified in the *interval* field. Similarly *call-duration* field tells engine about the minimum duration (in minutes) of the calls allowed for the organization in the time span equal to *interval*. These two fields are application to the total number of all the call types specified in

the *call-type* field. If the call duration and call frequency of all the monitored call types are less than the specified values, then the engine will not run the detection algorithm and thus avoids the unnecessary overhead from the engine.

ad-algo:
sensitivity: 1.3
adaptability: 0.25
interval: 10
threshold-restore: 'no'
call-freq: 10
call-duration: 10

The default *call-duration* values (in minutes) of each paid calls in the given interval. These values signify that the total duration of the corresponding call type is allowed, if the duration value is less than the specified value in the configuration value. These values are used to avoid the false positives during the office working hours.
call-duration:
mobile: 3600
premium: 3600
international: 3600

The *office-time* field tells what the office working hours are. This helps the engine to be more lenient to the traffic spikes during working hours. This helps in reducing the false positive rate. The working hours value for starting and ending time hours should be specified in 1-24 hour clock format.
office-time:
start_time: 8
end_time: 16

The *initial-timestamp* field tells the engine from which time, the engine will start the training of detection algorithm. If this field is not specified then, engine will fetch the first CDR record from the CDR database and will start the training from time stamp of the first CDR record.

The *training-period* field tells the engine for how long it should train the detection algorithm. The value should be specified in the configuration file in minutes and default value is 10800 (1 week). The *detection-start-ts* field tells the engine as from which time stamp, it should start the detection phase. This option is useful, when there is a time gap between the training phase and the detection phase time stamps. If the detection phase has to be started right after the training phase, then this option can be commented.
initial-timestamp: '2010-01-01 02:03:36'
training-period: 10800
detection-start-ts: '2010-01-11 00:00:00'

2 License

SipADE is available at http://software.uninett.no

References:

1. Sengar, Hemant and Wang, Haining and Wijesekera, Duminda and Jajodia, Sushil: *Detecting VoIP Floods Using the Hellinger Distance*, IEEE Trans. Parallel Distrib. Syst., 2008.

2. Van Jacobson, Michael J. Karels: *Congestion Avoidance and Control,* Proc. ACM SIGCOMM, November, 1988

3. The Hobbit Monitor: http://hobbitmon.sourceforge.net/

Appendix A

Installation Procedure

The SipADE engine requires **libpq-dev** and **libyaml-dev** libraries to build. The libpq-dev is used for the postgresql database communication and libyaml-dev is used for parsing the configuration file.

To install these libraries on you Ubuntu system, run the command as

$ sudo apt-get -y install libyaml-dev libpq-dev

This will install the required libraries for the SipADE. Now switch the SipADE command directory and run the command as

$ make

If no error has been reported during the build process, then to install the SipADE into the default location, run the command as

$ sudo make install

This will install the SipADE binary in to the /usr/local/sbin as "**sipade**" and store the default configuration file in to the /usr/local/etc/sipade with filename as "**sipade.yaml**". Now SipADE has been installed and ready to detect the anomalies, if any !!

To run the SipADE, it needs three postgresql databases: **CDR database, cdr_alert database and threshold database**. For the CDR database, the require fields are mentioned in the CDR module section. Make sure that you have all the required field in your CDR database. The SipADE will make the consecutive queries for all the records in the given interval, so to enhance the performance it is a good idea to create the index for calldate field too. To do so run the following command on the database command prompt:

create index ts_idx on cdr(calldate);

This will increase the performance of SipADE drastically, as SipADE will be able to fetch the data from the database by spending much less time in wait. This happens due to the creation of binary index instead of searching/fetching data serially.

For the cdr alert database, the database schema is given in the alert module, create the table by running the following command on the postgresql command prompt as:

```
CREATE TABLE cdr_alert (
    ser_id serial,
    alert_id integer DEFAULT 0 NOT NULL,
    cdr_id integer DEFAULT 0 NOT NULL,
    calldate timestamp with time zone DEFAULT now() NOT NULL,
    src text DEFAULT ' '::text NOT NULL,
    dst text DEFAULT ' '::text NOT NULL,
```

```
    billsec integer DEFAULT 0 NOT NULL,
    calltype text DEFAULT ' '::text NOT NULL,
    accountcode text DEFAULT ' '::text NOT NULL,
    PRIMARY KEY(ser_id)
);
```

This will create the cdr_alert data base. The last data base is to store the threshold values from each interval to make the restoration of the engine possible. To create the threshold database, run the command as:

```
CREATE TABLE threshold (
    threshold_id serial,
    num_int integer DEFAULT 0 NOT NULL,
    dur_int integer DEFAULT 0 NOT NULL,
    p_fint double precision DEFAULT 0.0 NOT NULL,
    p_dint double precision DEFAULT 0.0 NOT NULL,
    num_mob integer DEFAULT 0 NOT NULL,
    dur_mob integer DEFAULT 0 NOT NULL,
    p_fmob double precision DEFAULT 0.0 NOT NULL,
    p_dmob double precision DEFAULT 0.0 NOT NULL,
    num_prem integer DEFAULT 0 NOT NULL,
    dur_prem integer DEFAULT 0 NOT NULL,
    p_fprem double precision DEFAULT 0.0 NOT NULL,
    p_dprem double precision DEFAULT 0.0 NOT NULL,
    num_ser integer DEFAULT 0 NOT NULL,
    dur_ser integer DEFAULT 0 NOT NULL,
    p_fser double precision DEFAULT 0.0 NOT NULL,
    p_dser double precision DEFAULT 0.0 NOT NULL,
    num_dom integer DEFAULT 0 NOT NULL,
    dur_dom integer DEFAULT 0 NOT NULL,
    p_fdom double precision DEFAULT 0.0 NOT NULL,
    p_ddom double precision DEFAULT 0.0 NOT NULL,
    num_emr integer DEFAULT 0 NOT NULL,
    dur_emr integer DEFAULT 0 NOT NULL,
    p_femr double precision DEFAULT 0.0 NOT NULL,
    p_demr double precision DEFAULT 0.0 NOT NULL,
    num_total bigint DEFAULT 0 NOT NULL,
    dur_total bigint DEFAULT 0 NOT NULL,
    dist_value double precision DEFAULT 0.0 NOT NULL,
    mean_dev double precision DEFAULT 0.0 NOT NULL,
    threshold double precision DEFAULT 0.0 NOT NULL,
    last_ts timestamp with time zone DEFAULT now() NOT NULL,
    PRIMARY KEY(threshold_id)
);
```

Now as all the required databases are in place, to run the SipAD engine type the command as

```
$ sipade
```

This will start the engine and you will see the output as

[25/5/2010 -- 09:41:48] <INFO> Training the engine for detection of anomalous behavior...

[25/5/2010 -- 09:41:49] <INFO> SIP Anomaly Detection Engine has been started successfully...

These messages shows that the engine has been started successfully and been in place to detect the anomalies in the call pattern. You can also specify the different configuration file by passing the file path with *-c* parameter as.

$ sipade -c <path to configuration file>

To generate the documentation for the SipADE code, switch to the *docs* directory in the sipad source folder and run the command as

$ doxygen sipade_docs.conf

The above command needs doxygen program to be installed on your system to work. This will generate the *htm* files in the *doxygen* folder under the *docs* directory. Open the *index.htm* file, this will have the links to all the remaining file codes and functions. It is a good way to go through the different functions and their call graphs to get the overview of the code and its working.

If you have encountered any error, then please check if you have setup the CDR database properly and specified the connection information for all the databases correctly in the configuration file. The call type names are case sensitive, so be careful while typing them in the configuration file and also indexing them for CDR database.