oTN-2009-01 openlab Summer Student Report

Development of a Security Tool for Oracle Databases

Jimenez, Raul
Girone, Maria & Canali, Luca

26 of August 2009
Version 1
Distribution:: **Public**

Abstract

The goal of this openlab summer student project was to increase the security of the CERN physics Oracle databases. The way to achieve this is via a tool to automatically perform various security tests on the databases. The proposed way is focused in Oracle Databases Auditing. The tool will collect auditing information along databases. The auditing trails will be analyzed to obtain a response about what is happening into the databases.

1 Auditing Tool

1.1 Auditing is necessary

Auditing is a necessary activity that has to be performed in any Oracle security project because it is the most effective tools in fight intrusions and wrongdoing. An audit trail is very effective to help administrators not only to stop people from doing wrong things but to alert then on suspicious activity and for doing an investigation to find out what went wrong in critical process.

Today's organizations have increasingly become dependent on IT applications that allow users to access confidential information. Critical functions are dependent of this IT applications as well as there is exponential increase in vulnerabilities. As everybody knows, CERN is under permanent attack, and new vulnerabilities are discovered frequently. In this context, to secure and audit physics databases is a necessary practice.

1.2 Components

The Auditing tool has been designed in three different parts. Those parts are the extraction process, the Auditing Engine and the Monitoring tool. The Auditmon Engine is part of the project but has been developed by CERN staff.

- Extraction Process.
  Auditmon is a <u>new component</u> which is responsible of extracting audit information from the system tables. The auditing process is formed by three different procedures written in a PL/SQL for simplicity and maintenance reasons.

- Auditmon Engine.
  It is an Oracle Package loaded on a dedicated Database. It is a rule based module running after data collection to discover improper actions or irregularities in DB access. This module is part of the Auditing Tool and it was developed by other workmate. (Katarzyna Dziedziniewicz). In case of simple rules each predefined rule is executed on data from one or more databases and when the result is above the threshold defined it is recorded in the result tables with the severity level: info, warning, alert. The aggregate/complex rules are applied to discover irregularities over time or repeating in data from different DBs (failed logons over time, access from unknown network to different DBs).

- Monitoring Tool.
  Last task consisted in preparing a Monitoring Tool to present the information generated by the Audit Engine. This software use the auditmon schema to present several reports about the security problems found by the Auditing Engine.

These three components must be integrated. The components will run independently but using the Auditmon schema.

2 Extraction Process

2.1 Auditing Extraction Procedure

Three different processes have been prepared to transfer the Audit trails from oracle databases.

- AuditExtration
- AuditTabPartitioner
- dblinkLoop

The auditing process is a stored procedure written in a PL/SQL, which offers several distinct advantages like allowing good maintenance. By running the procedure on a dedicated database it extracts audit information from the system tables. Another procedure (AuditTabPartitioner) prepares the table partitions with a specific name format. The last one (dblinkLoop) checks the databases list, and it runs the main procedure each time on a different ddbb.

The extraction process will create different tables (one per database) to insert the data into the destination database. The procedure connects to the other databases via dblink that should be available during the extraction process. So finally there is a dedicated database which contains the auditing procedure, the dblinks connections and the auditing destination tables to hold the information.

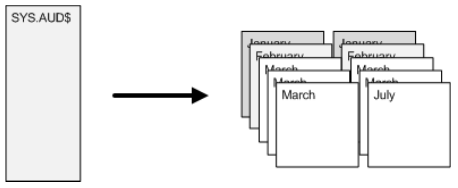
There are many auditing options to collect on each database depending on the audit policy, but only the following data is recorded on the destination tables.

Source	Destination	Description
spare1	OS_USERNAME	Operating system login username of the user whose actions were audited.
userid	USERNAME	Name (not ID number) of the user whose actions were audited.
userhost	USERHOST	Client host machine name.
terminal	TERMINAL	Identifier of the user's terminal.
ntimestamp#	TIMESTAMP	Date and time of the creation of the audit trail entry (date and time of user login for entries created by AUDIT SESSION) in the local database session time zone.
act.name	ACTION_NAME	Name of the action type corresponding to the numeric code in the ACTION column.
Logoff$time	LOGOFF_TIME	Date and time of user log off.
Logoff$lread	LOGOFF_LREAD	Logical reads for the session.
logoff$pread	LOGOFF_PREAD	Physical reads for the session.
logoff$lwrite	LOGOFF_LWRITE	Logical writes for the session.
Logoff$dead	LOGOFF_DLOCK	Deadlocks detected during the session.
sessionid	SESSIONID	Numeric ID for each Oracle session.
returncode	RETURNCODE	Oracle error code generated by the action.
clientid	CLIENT_ID	Client identifier in each Oracle session.
sessioncpu	SESSION_CPU	Amount of CPU time used by each Oracle session.
Proxy$sid	PROXY_SESSIONID	Proxy session serial number, if an enterprise user has logged in through the proxy mechanism.
User$guid	GLOBAL_UID	Global user identifier for the user, if the user has logged in as an enterprise user.
instance#	INSTANCE_NUMBER	Instance number as specified by the INSTANCE_NUMBER initialization parameter.
process#	OS_PROCESS	Operating System process identifier of the Oracle process.

Structure of the Main Procedure – Audit Extraction

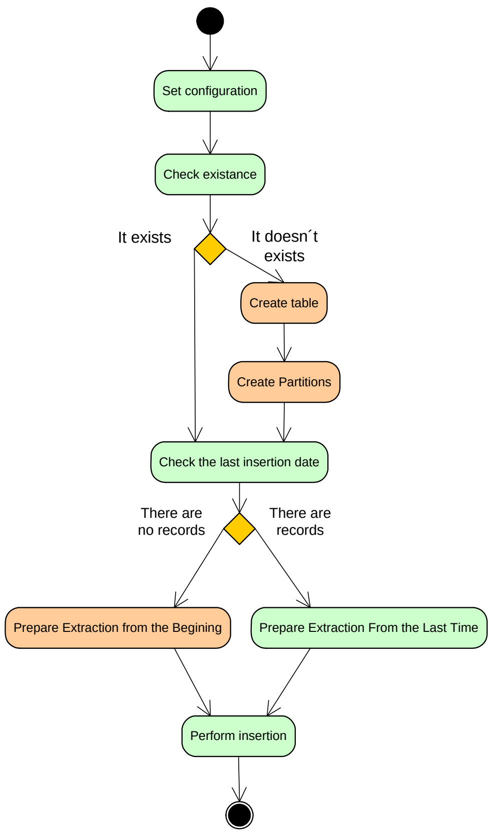
Audit extraction procedure is in charge of extracting the audit information from a list a databases. This procedure has two parameters which are used to build a query using different dblink connections every time the procedure is invoked. The procedure checks if the destination table exists and it tries to create partitions if it is needed.

Partitioning is the ability of the database to take very large tables or indexes and physically break them into smaller and more manageable pieces. Parallel processing can be useful to break a large process into smaller parts that can be processed independently. Partitioning can also be used to break a very large table in several parts that can be independently managed.



The destination tables are supposed to be large tables, and a big amount of data is going to be transferred into it, so having a partitioned table will facilitate the operations. As the procedure is used in parallel, this solution will potentially reduce the amount of time the operation takes. This adopted solution (partitioning) also increases availability when datafile goes bad. Using partitioning in separate tablespaces, only one partition would be affected.

At this point, the procedure is prepared to create new partitions on demand each month. It means that tables are partitioned by a timestamp field. A new partition will be created every month and it will contain the database audit trail for that month. It has been prepared in this way because partitioning must be applied in such a way as to solve a real problem, and tables should be separated in a logical way to take advantage of the performance improvement that partitioning brings. The only problem is if somebody else tries to create the partitions on a wrong date or to use a different name format, then errors may appear.
To avoid this kind of error, it calls another procedure to prepare partitions with a normalized name format. This procedure prepares the table to hold information for the next two years.



It is necessary to consult the databases table to know about the availability of databases and dblinks connections. The database names are used as parameters on the main procedure. So for this reason there is a procedure that is in charge of looping into the databases list. A cursor is prepared to loop into the list and an instance of the main procedure is called to extract the audit data every time.

2.2 Problems during the development process Performance

One of the problems encountered during the development process was related to the Databases Time Zone configuration. The main procedure is prepared to extract information from databases using a timestamp field. The last insertion date is checked before performing the extraction, and then it is used to extract information from that date to the current date. The time zone configuration of Test1 ddbb is different from the rest of databases, and the number of records transferred was wrong during the test period.

To be exact, the time zone of “Test1” database is two hours less than the rest. Every time the procedure ran more records than intended were transferred.

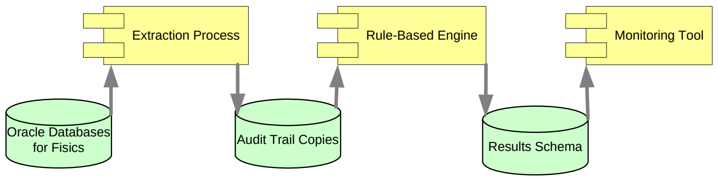
The date transferred is exactly the same date as in the source database. The problem consists in how oracle interprets the timestamps using the local time zone representation. This timestamp is necessary to compare and to extract recent records.

Several different methods were tried to fix this problem. Many functions were used to avoid the internal timestamp conversion, and finally the solution resides in using the UTC time function. Thanks to this function, it was possible to perform a successful comparison and to finish the extraction procedure.

2.3 Integration process

The last step consists on integrating the extraction procedure with the Auditing Engine to make them work together. As it was described above, a Scheduled procedure launches the extraction job and to record information into the partitioned tables. The engine also counts with a scheduled procedure that reads the schema and starts the analysis process. Both jobs have been scheduled to run every hour with a difference of half an hour to avoid overlapping the extractions.

The scheduled procedure was modified to carry out a set of insertions in a new table. This table holds information about the executed jobs. The same idea could be extended to the Auditmon Engine in order to know the processes status. As additional work, this information could be displayed on the monitoring tool to let know the user when was the last execution and if the information is up to date.



3 Monitoring Tool.

3.1 Previous Considerations

After finishing the Auditing process, the next task consisted in preparing a Monitoring Tool to present the information generated by the Audit Engine.

Many options were taken into consideration:

- Integrate the results schema into the Racmon Interface.
- Look for Monitoring Software to present results.
- Start a new development project in a known technology to properly integrate into the project.

First, the Racmon interface was considered as an alternative to continue the development and to add new features to present the auditing results. This option was ruled out because the code was not documented enough and it would take more time to understand the code than start a new web application.

Another option was to research available monitoring tools. A powerful tool was tested for automatic reports generation (Jasper Report Software). It was discarded because it needs a complicated deployment and this tool is not customizable at all.

The adopted solution was to start a new project as a web interface using PHP. Using PHP language was a maintenance decision and this language was also used to build other monitoring tools to present security information. Furthermore, using web interfaces makes the integration between them easier. Just by using link tags on the views it can be simply integrated.

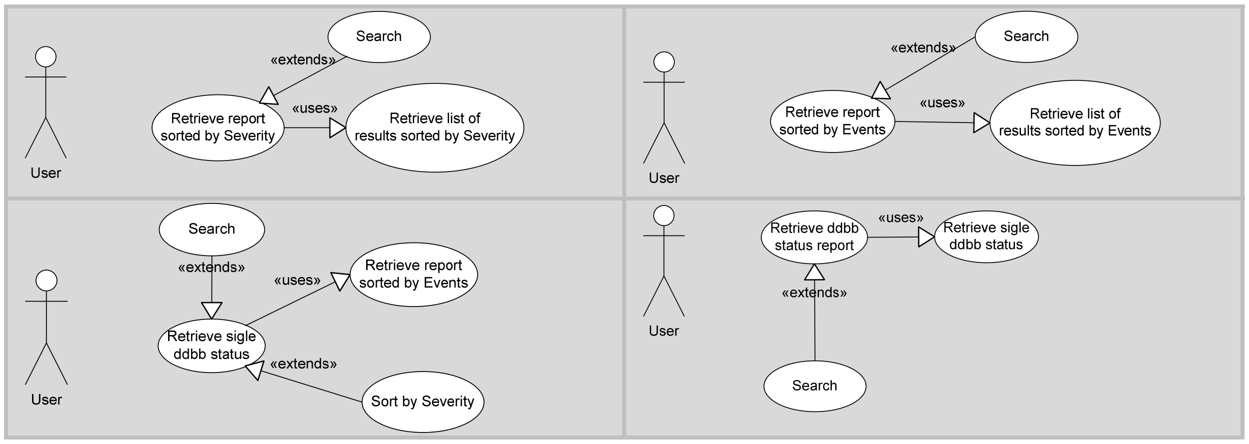
3.2 Interface Requirements.

The information recovered from the Auditmon Schema must be presented in different ways to allow the user to easily find the information he needs. A basic set of requirements that the interface must achieve were described. After implementing the basic requirements, it was easy to add many other features because of the very organized web architecture.

List of the main operations:

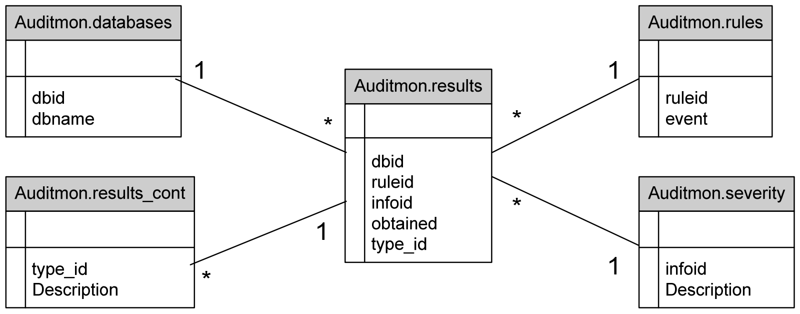
- Show a global report about events on the systems.
- Show a global report about the severity of the events.
- Show a more detailed report about events in each database.
- Prepare the view to sort it by Severity.

Uses cases diagrams explain the interface requirements:


User
Search
«extends»
Retrieve report sorted by Severity
«uses»
Retrieve list of results sorted by Severity
User
Search
«extends»
Retrieve report sorted by Events
«uses»
Retrieve list of results sorted by Events
User
Search
«extends»
«uses»
Retrieve report sorted by Events
Retrieve sigle ddbb status
«extends»
Sort by Severity
User
Retrieve ddbb status report
«uses»
Retrieve sigle ddbb status
«extends»
Search

3.3 Operations

The information that the application must present is stored in the Auditmon schema. Those operations are directly translated to SQL queries. The database schema is as follows:



A set of SQL queries describes what the view has to show. Those SQL queries are also used for testing the application.

3.4 Using Designs Patterns.

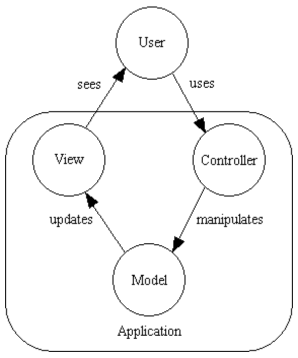
Many designs patterns have been adapted in order to build a maintainable application and to allow the programmer to add new features.

Dispatcher

The Dispatcher design pattern introduces an intermediate layer between clients and servers, the dispatcher component. It provides transparency of a name service, and hides the details of the establishment of the communication connection between clients and servers.
Dispatcher Component allows adds a new empty view just by typing a new line in a "switch case".

Model-View-Controller

The Model-View-Controller (MVC) pattern divides an application into three components.

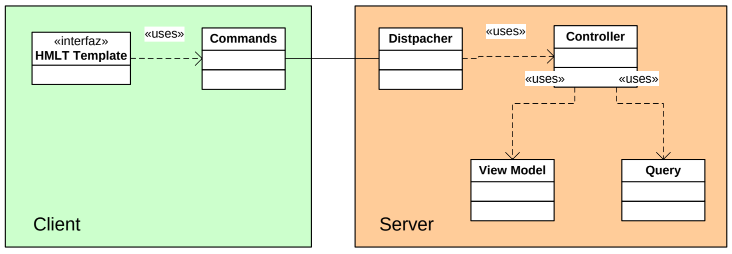


Input --> Processing --> Output
Controller --> Model --> View

- The **model** contains the core functionality and data. It is independent of specific output representations or input behaviour.
- The **view** displays information to the user. A view obtains data from the model. There can be multiple views of the model.
- The **controller** handles user input.

- NOTE: In this application the model concept refers to the html tables and the information displayed but not to specifics data types. The application use sets of data retrieved directly from the database.

Next picture show a brief idea of design:



3.5 Client Side

The user can find many options on the interface to retrieve the Auditing Information. When the URL is accessed, a set of JavaScript functions are loaded to support the user operations. The functionalities of those functions are multiple. As it was mentioned in the design section, the application follows the Command / Dispatcher Pattern by including a JavaScript function which sets some hidden input variables. Those input variables are sent to the server side to decide what operation the user wants to execute. The same URL is accessed each time the user clicks on a link, and the Dispatcher switches the requested action to pass the control to the rest of the components.

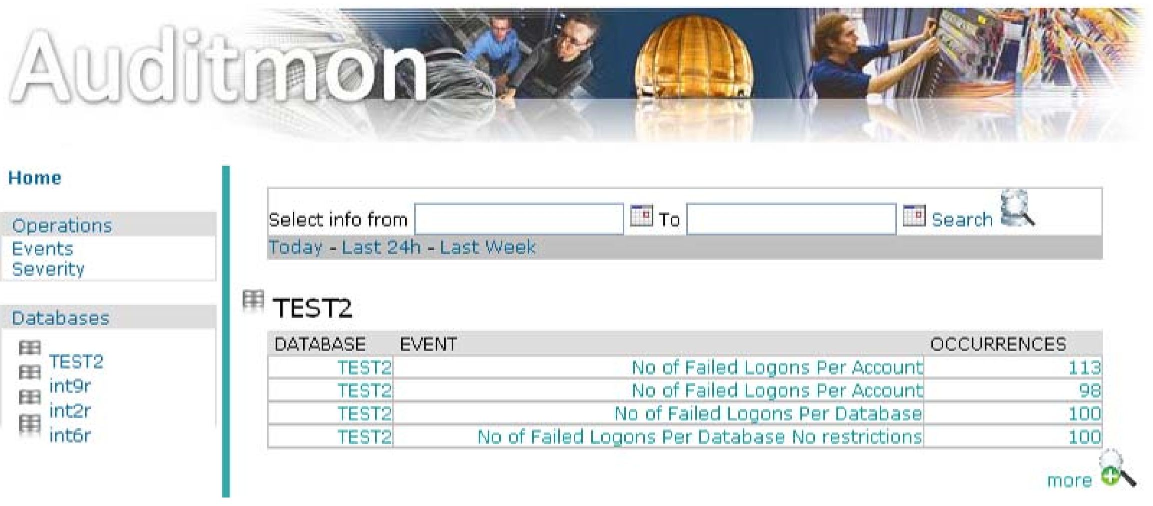
3.6 Server Side

The main component is located on control.php. The hidden variables are read on the server side to carry out the requested operation and to prepare the response. Those variables are sent via POST by a JavaScript function called “redir2(<parameters>)”. Once the POST variables are accessed, a large function redirects the request to the appropriate control code. In those sections many operations are executed before displaying the view. Following a MVC schema-based, we can find several database operations in the control sections, as well as several building-view functions depending on what command was sent.

3.7 Views

As it was mentioned above, the application uses an html template which is filled depending on the command. The main view is the index.php which contains the main html and many other resources as the JavaScript’s. The controller component is included to display the information generated by the command. It intends to separate the html from the PHP code to obtain a readable code and to easily understand how the display was generated. For this reason there is another file which contains a set of functions to prepare pieces of html code based on input parameters. It is in the control section where the information is retrieved from the database and then passed to the display functions. At the end of the execution a variable which contains all the html code gets ready to be displayed.

In few cases it is necessary to retrieve information directly from the views functions because some asynchronous operations need to connect to ddbb and to generate html code pieces without following the dispatcher schema.



DATABASE	EVENT	OCCURRENCES
TEST2	No of Failed Logons Per Account	113
TEST2	No of Failed Logons Per Account	98
TEST2	No of Failed Logons Per Database	100
TEST2	No of Failed Logons Per Database No restrictions	100

Ajax Queries

Ajax is a group of web development techniques used on the client-side to retrieve data from the server asynchronously in the background without interfering in the behaviour of the existing page. Using Ajax methods allows the construction of dynamic interfaces.
The tool has been designed not only to show a short report about the security status but to directly retrieve detailed information about the events that has occurred on the selected dates.

Thanks to an Ajax component, it is possible to extend the information to the screen without refreshing the view. This component is an open source engine that automatically generates a set of JavaScript functions on the client side, which allows the application to perform asynchronous operations on the server side. A PHP module is in charge of receiving the requests. This module is ready to invoke functions and it must to be previously registered on the system. Each function performs the appropriate operations as the control sections do to return a new piece of html code.

The view must be prepared previously to receive the information by Ajax methods. Additional information is transferred on demand when required. This solution also produces a light interface allowing the user to expand the information when it is necessary.

Steps to integrate the Ajax library (xajax 0.5)

1) The xajax library must be included on the code.

```
require_once("ajax/xajax_core/xajax.inc.php");
```
Index.php

2) Instantiate the object:

```
$xajax = new xajax();
```
Index.php

3) Redirect requests:

```
$xajax->setRequestURI("request_actions.php");
```
Index.php

4) Register Functions:

```
$xajax->registerFunction("db_rule_results_details")              ;
$xajax->registerFunction("db_severity_details")     ;
```
Index.php

5) Create the response:

```
        function db_rule_results_details($div_id, $datefrom, $dateto, $database, $ruleid)
        {
                // OUTPUT:
                $newContent = single_detail_table($datefrom, $dateto, $database, $ruleid);

                // Instantiate the xajaxResponse object
                $objResponse = new xajaxResponse();

                $objResponse->assign($div_id, "innerHTML", $newContent);

                //return the  xajaxResponse object
                return $objResponse;
        }
```
Request_action.php

6) Generate JavaScript functions:

```
        <?php $xajax->printJavascript(); ?>
```
Index.php

4. Conclusions

The result of this work is a tool that allows administrators to conduct a follow-up of databases using the reports generated by the Monitoring tool. The proposed tasks have been accomplished including additional features that make it easy to be managed. The collection of the auditing trails also means that new features could be implemented to look for new risks.

Thanks to the team atmosphere and the group work of the IT-DM section is was easy to manage my work as an OpenLab summer student. It has been an excellent experience learning best practices of Oracle Databases under the supervision of CERN experts.

5. Possible Improvements

- The process responsible for running the Extraction procedure also records its status in a table. The software administrator can check the results to know how the transfer process was performed. This feature could be also implemented into the Engine Package to make aware administrators that the task has been performed.

- Additional view on the monitoring tool to display the status of the jobs, so the user would know when the information is updated.

- The Monitoring tool can be prepared to run and modify the Engine and process parameters in order to make a customizable system.

6. Bibliography

[1] Effective Oracle by Design (Osborne ORACLE Press Series).

[2] Oracle Documentation. Oracle Database 11g Release 1 (11.1) Documentation.

[3] Design Patterns – A personal perspective By Tony Marston.
http://www.tonymarston.net/php-mysql/design-patterns.html

[4] www.Javascriptkit.com. - Date Time Picker

[5] http://www.php.net/

[6] The Apache Software Foundation.
http://www.apache.org/

[7] xajax PHP Class Library- The easiest way to develop asynchronous Ajax applications.
http://xajaxproject.org/

7. Appendix

7.1 Appendix A: List of functions and components

- index.php
  - Html template.
  - CSS.
    - Cern.css
    - Table.css
  - JavaScript functions.
    - function redir2(request_, view_, selection_, time_)
    - function moreinfo(element)
    - function moreinfobyid(id)
    - function selecttoday(id_1, id_2, date_from, date_to)
- controller.php
  - Dispatcher.
    - function switcher($req)
- connexion.php
  - OCI driver.
    - function load_sql()
    - function connection()
    - function extraction($conn, $sql)
    - function matrix_extraction($conn, $sql)
  - Database Queries.
    - function prepare_db_menu()
    - function single_db_rules($database)
    - function global_db_info_rules($from, $to)
    - function global_db_severity_info($from, $to)
    - function single_db_severity_info($database, $infoid, $from , $to)
    - function db_results_details($from, $to, $database, $ruleid)
    - function db_results($from, $to, $database)
  - Other functions
    - function parse_date($date)
- models.php
  - Html data models.
    - function single_db_view($db, $table1, $table2)
    - function show_more_info_from_rule($name, $component_id, $database, ...)
    - function show_severity_info($name, $component_id, $database, $infoid, $from , $to)
    - function print_test_table($title, $head, $content, $foot)
    - function print_link_table($title, $content, $from, $to)
    - function single_detail_table($from, $to, $database, $rule)
    - function print_db_menu($title)
    - function global_information_view($data, $title, $from, $to)
    - function global_information_by_severity_view($data,$title, $from, $to)
    - function calendar_view($id)
    - function calendar_panel_view($id_from, $id_to, $message, $goto, $selection)
  - Other functions.
    - function more_info_table($component_id)
    - function LastWeek()
    - function image_($image)
    - function space_($n)
    - function separator_()
- request_actions.php
  - Ajax queries.
    - function db_rule_results_details($div_id, $datefrom, $dateto, $database, $ruleid)
    - function db_severity_details($div_id, $datefrom, $dateto, $database, $infoid)
- External Components
  - Ajax library.
  - Calendar component. (JavaScript).

7.2 Appendix B: How to add new features

1) Add a new case in the main function (dispatcher) in “control.php”.

```
function switcher($req) {
...
        case 'showsingledb': break;
```

Controller.php

2) Add an html link using the function “Redir2” as action to send the request.

```
<a href="javascript: redir2('showsingledb', 'None', 'None', 'None'); document.request_form.submit();"> ...</a>
```

Index.php

3) Create a query function in “Connection.php”.

```
function single_db_rules($database){
        $db = connection();
        $query = "       SELECT    r.DESCRIPTION                         Event              ,
                                            COALESCE(amounts.n, 0)  Occurrences
                              FROM    rules     r
...
```

Connection.php

4) Prepare a view function in “models.php” or use an existing function that loops into the results.

```
function print_test_table($title, $head, $content, $foot) {
...
        foreach ($content as $key => $val) {
                $document.= "       <td height=15 style='border-left: 1px solid #DDDDDD; border-right: 1px solid ...
```

Models.php

5) Prepare the “Case clause” of the controller procedure to retrieve the information and print the view.

```
case 'global':
// OUTPUT:
print calendar_panel_view('INPUT_FROM', 'INPUT_TO', $message, 'global', null);        // prepare the date input components.

$data = global_db_info_rules($date_from, $date_to);                                      // retrieve information from the database.

// OUTPUT:
print global_information_view($data, "Events", $date_from, $date_to);                   // prepare the view.
break;
```

Models.php

7.3 Appendix C: SQL queries for testing.

```
/*****************************************************************************/
/* The next query is executed when a single database report is requested.    */
/* Single database view sort by events.                                      */
/*****************************************************************************/


SELECT      d.dbname            database,
            r.ruleid                    ,
            r.DESCRIPTION       Event   ,
            COALESCE(amounts.n, 0)  Occurrences
FROM    rules       r
             CROSS JOIN
        databases   d
             LEFT JOIN
        (
             SELECT    re.dbid   dbid    ,
                       re.ruleid ruleid  ,
                       count (*) n
             FROM    results   re
             WHERE re.OBTAINED
                    BETWEEN to_timestamp('24-Aug-2009 00:00:00', 'DD-MM-YYYY HH24:MI:SS')
                        AND to_timestamp('24-Aug-2009 12:8:25', 'DD-MM-YYYY HH24:MI:SS')
             GROUP BY        re.dbid   ,
                             re.ruleid
        ) amounts   ON  r.ruleid  = amounts.ruleid
                    AND d.dbid    = amounts.dbid
WHERE d.dbname = 'TEST2'
ORDER BY d.dbname, r.description;
```

```
/* ***************************************************************************/
/* The next query is executed to show results sorted by severity.            */
/* Both queries must return exactly the same number of registers.            */
/* ***************************************************************************/

SELECT      d.dbname                Database,
            r.INFO                  Info    ,
            COALESCE(amounts.n, 0)  Ocurrences
FROM    info_type     r
          CROSS JOIN
        databases   d
          LEFT JOIN
        (
        SELECT    re.dbid   dbid    ,
                  re.infoid infoid  ,
                  count (*) n
        FROM    results   re
        WHERE re.OBTAINED
              BETWEEN to_timestamp('24-Aug-2009 00:00:00', 'DD-MM-YYYY HH24:MI:SS')
                  AND to_timestamp('24-Aug-2009 12:8:25', 'DD-MM-YYYY HH24:MI:SS')
        GROUP BY  re.dbid   ,
                  re.infoid
        ) amounts   ON  r.infoid  = amounts.infoid
                    AND d.dbid    = amounts.dbid
WHERE (1 = 1)
   AND d.dbname = 'TEST2'
ORDER BY d.dbname, r.INFO
```

```
/*****************************************************************************/
/* The ajax querie to look for details of the main table.                   */
/* In this case the number of results could be not the same because the    */
/* multiplicity of those tables are one to many. One result contain many    */
/* details                                                                  */
/*****************************************************************************/
select          d.dbname        Database,
                r.OBTAINED      Obtained,
                rc."content"    Content ,
                info.info     Info
FROM    results       r
            JOIN
        databases         d     ON r.dbid = d.dbid
            JOIN
        rules             rul   ON r.ruleid = rul.ruleid
            JOIN
        info_type         info  ON r.infoid = info.infoid
            LEFT JOIN
        result_cont   rc        ON  r.resultid = rc.resultid
WHERE rul.ruleid = 2
  and d.dbname = 'TEST2'
  and r.OBTAINED
       BETWEEN to_timestamp('24-Aug-2009 00:00:00', 'DD-MM-YYYY HH24:MI:SS')
           AND to_timestamp('24-Aug-2009 12:8:25', 'DD-MM-YYYY HH24:MI:SS')
ORDER BY d.dbname, r.OBTAINED



/* **************************************************************************/
/* The next query is to retrieve the full report sorted by events.          */
/* **************************************************************************/
select r.description      event ,
       r.ruleid           ruleid,
       COALESCE(amounts.n, 0) N
from    rules     r
             LEFT JOIN
        (
         SELECT  count(*)    n,
                 res.ruleid  ruleid
         FROM results res
         WHERE res.OBTAINED
               BETWEEN to_timestamp('23-Aug-2009 01:08:07', 'DD-MM-YYYY HH24:MI:SS')
                   AND to_timestamp('24-Aug-2009 01:08:07', 'DD-MM-YYYY HH24:MI:SS')
         GROUP BY (res.ruleid)
         ) amounts
         ON r.ruleid = amounts.ruleid



/* **************************************************************************/
/* The next query is to retrieve the full report sorted by Severity         */
/* **************************************************************************/
select     i.info               SEVERITY        ,
           i.infoid             INFOID          ,
           COALESCE(amounts.n,0)   N
from    info_type     i
             LEFT JOIN
        (
        SELECT  count(*)    n,
                res.infoid  infoid
        FROM results res
        WHERE res.OBTAINED
              BETWEEN to_timestamp('23-Aug-2009 01:08:07', 'DD-MM-YYYY HH24:MI:SS')
                  AND to_timestamp('24-Aug-2009 01:08:07', 'DD-MM-YYYY HH24:MI:SS')
        GROUP BY (res.infoid)
        ) amounts
        ON i.infoid = amounts.infoid
```