--- Final Project Studies ---

Intelligent Network Management Framework (INMF)

Created the :	07/07/2004
By :	Guillaume Andreys
Contact :	guillaume.andreys at laposte.net
Version :	1.0
Last modified :	02/08/04

Abstract :
In reference to the project presented in the paper "Intelligent real-time reactive network management", this report present the specification of the produced framework, and the documentation of the installation and configuration.

Thinks to Abhishek Jain for is contribution on the Tools and the scripts.
Thinks to G. Sivakumar and Frederique Biennier to guide me on this project.

Table of Contents

1 Introduction

This report is in the context of my Final Studies Project. This project took place in IIT (Indian Institute of Technologie), Mumbai, India, from April to August. Before that, I also did some work in France, especially the study of intrusion detection techniques [Doc4]and the study of the OSSIM [Tool2] project.

Network monitoring and security is always a great problem, and the importance of such activity is growing with the augmentation of the use of Internet.

There is a lot of tools which are very powerful. We can classify it in different category :

- The intrusion detection systems : Those tools are design for network security. They are collecting information on the network to detect intrusion signature. The problem of such system are false alarms, the hight system resource that it is using and a lack of communication with other components of the network.
- The network management systems : A lots of tools exist to monitor various data on the network. It can be only getting SNMP informations, or analysing logs, complete solution of network monitoring ...
- Transversal tools : Some tools can get informations from various point. Some just provide a central point where all data are collected, and others are more intelligent and can perform correlation to avoid false alarms.

But in all that collection of tools, usually, either it need to run continuously, either ones should launch it manually. It imply that network management and security become something very costly, either in material resources, or in humane resources (time). Indeed, providing a good security imply doing detail analysis to be able to get all needed informations all the time.

The aim of this project is to fill this lack by providing a framework who will allow the user to automatise the actions of launching and stopping tools and scripts on a network. We want to provide an « *an intelligent, automatic, real-time reactive, sophisticated layered ('lazy', one may say) network analysis* ». Automatic because one of the principal aim is to avoid the administrators to have to manually launch tools or scripts in case of problems. Real-time because we will react immediately. Intelligent because we will allow the user to define rules and security policy to define what to do and when to do it. And layered network analysis because we will start with hight level (aggregated) network informations to go to details informations.

We will call this project : Intelligent Network Management Framework (INMF). This name doesn' tcover every functionalities but is short and easy to remember.

This document will concentrate on the elaboration of the framework, the choice of the tools and the associated documentation.

2 Requirement

This project should be combination of a framework and tools (existing open source products and/or new tools) that are integrated to provide an infrastructure for network monitoring and internal security.

The name of the project is **I**ntelligent **N**etwork **M**anagement **F**ramework (INMF).

The main point is performing a top to button analysis, gradually increasing the collected data. We started from aggregated data (like main switch bandwidth information), to eventually (if necessary) figure out users data.

Various points in the network are monitoring, using a less resource consuming tool and on receiving the first alert (anomaly) at the highest level, further analysis is done which form the inner layers of the overall process.

Here are the basics requirements :

- Tools to **perform the hight level data collection and anomaly detection** with low resource consuming.
- Tools and scripts to **collect some particular informations** like user name, user IP ... We will always try to find the less resource consuming method, according to the informations we want to get.
- A framework which will allow use to :
  - **Collect data from the tools** wherever it is on the network.
  - **Performing a decision process**. The decision process should take into account some user' sdefined rules (what to do), priority settings (priority on the user/user group, IP/network and time) and collected data.
  - **Launching/stopping tools**, with different parameters, wherever it is installed on the network.
  - Helping the user to **write rules and defining a security policy**.
  - Doing **alert reporting**.

Some constraints :

- **Open source** : We choose to make an open source project. This implies the use of open source component or tools.
- **Linux compatibility** : This project must be compatible with Linux.
- **Real-Time :** We must have an almost real-time analysis (in the scale of network analysis). When we get an hight level alert, we need to immediately react if we want to go into details.
- **Performance** : One of the main point of this project is to decrease the resources used by the network analysis tools. Then our framework shouldn't used those released

resources.

- **Maintainability :** Has this project will be Open Source, a large number of developer can have some participation on it. The software should be easy to maintain.
- **Flexibility :** To assure some diffusion of the project, it should be flexible enough to allow adaptation to other case than the initial case study presented on this report.
- **User friendly :** For the first version of the framework we won't offer much user friendly functionalities, but in the conception we have to think of the future and take this constraint into account.

3 Framework specification

3.1 Global architecture

The framework should perform :

- **The data collection** : On each host where there is tools which are collecting informations, an host based agent should collect those data.
- **Launching/Stopping tools** : As one off the main aim is to save resources, tools are not running all the time. We need to start and stop them. We will integrate this functionalities in the same agent of the data collection as we need those two functionalities on each host.
- **Decision process** : To perform the decision process we need some user's defined rules and policy and all the collected data. Indeed, we need a central point where all collected data will be send. We will call it the Manager.
- **Alert reporting and rules/policy writing** : This should be located on the Manager.

This lead us to a distributed architecture, where agents on many hosts will send informations on a central server. The server should also send data (decision result) to the agents. All this communication will be a network communication.

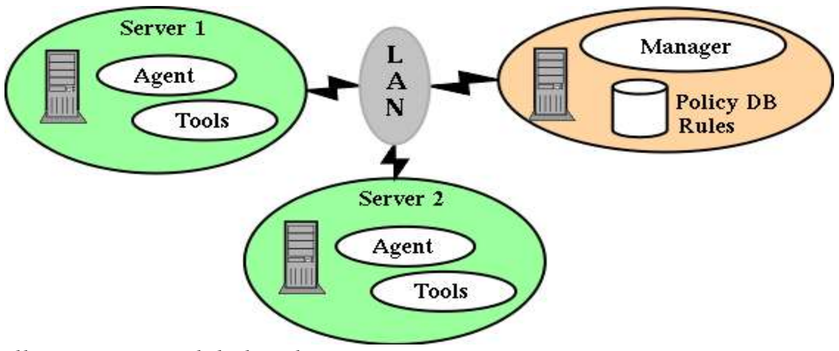


Illustration 1 : Global architecture

3.2 Agents

3.3 Functionalities

As said before, the Agents are located on hosts where there is network management tools.

- ***Data collection function*** : It collect data from the tools. Collected information can be alarms, network data, users data... As sources and format can be different, we need to normalize the information. Moreover, we have to find some solution to make it easy to add new supported tool to the framework. We can't afford writing an entire agent for every new tool.

  We choose the solution of using plugins. Each plugin is write for a specific application or script. It will read specific data of the tool (the method to accede data depend of the tool), put those data in a normalized form and transmit it to the Agent. We choose the following convention for the normalized form : `DataName:DataValue\n`

  Where DataName is an arbitrary name of the data we are collecting, and DataValue the corresponding value. On the new line (after \n) we can send an other couple DataName, DataValue for the same tool.

  For those plugins, we choose the perl language. It is very powerful for pattern matching, the main work of the plugins.

  We can't afford collecting data continuously, we chose to collect it regularly, on a defined interval of time. We will launch all plugins corresponding to running tools, and getting the data collected by the plugins.

- ***Sending data to the manager*** : If we have collected data from tools, we need to send it to the manager. It act as a server, and is waiting for connections from the agent. The agent, then, can connect to the server (it need to know the server IP and port) and send all the data it has. The message is detailed in Communication Protocol.

- **Receiving orders from the manager** : When the manager take a decision, it will send it to the concerned agent. This time, the agent act as a server, and wait for a connection from the manager. The received message is detailed in Communication Protocol.

- **Launching/Stopping tools** : From a manager order, we need to launch or stop tools. For this we need to know the command to launch and stop the tool. Some tools need to be launched by default, at the agent start. We must be able to give parameters to the tools when we launch it.

3.3.1 Implementation choices

Looking at the required functionalities, we can figure out that we will need two different thread : One will wait for a manager connection and perform actions on the tools (server thread). The other is reading tools data via plugins and sending it to the manager (toolReader thread).

The only information which is shared by the two thread is the state of the tools (stopped or running) : The server thread need to update information when it start or stop a tool, the toolReader thread need to read which tool are running to launch the appropriate plugins. We will used a shared memory (POSIX object) protected by a mutex.

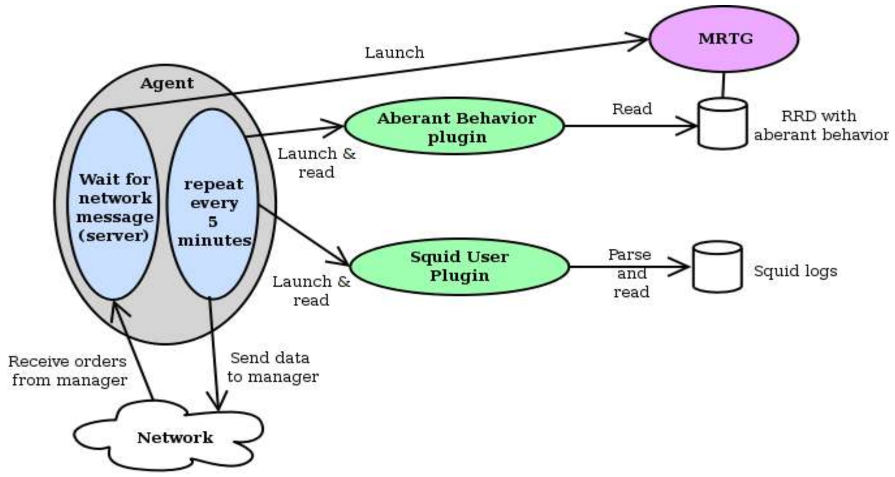


Illustration 2Agent's operations

We choose to write the framework in C++. This for performance question, maintainability (object language) and multi-threads capability.

To make the development quicker, we will use some existing library and class :

- To manage the configuration file, we decide to use XML format and the PACC::XML [Tool3] set of class. This set of class will allow us to parse an XML file, finding nodes, getting attributes ...
- To manage the network server and client side, we will use the PACC::Socket [Tool3] set of class. This set of class implement a TCP client and a TCP Server. The TCP server is multi-thread (one thread per connection).
- To manage the logs we will use the log4cpp [Tool4]library.

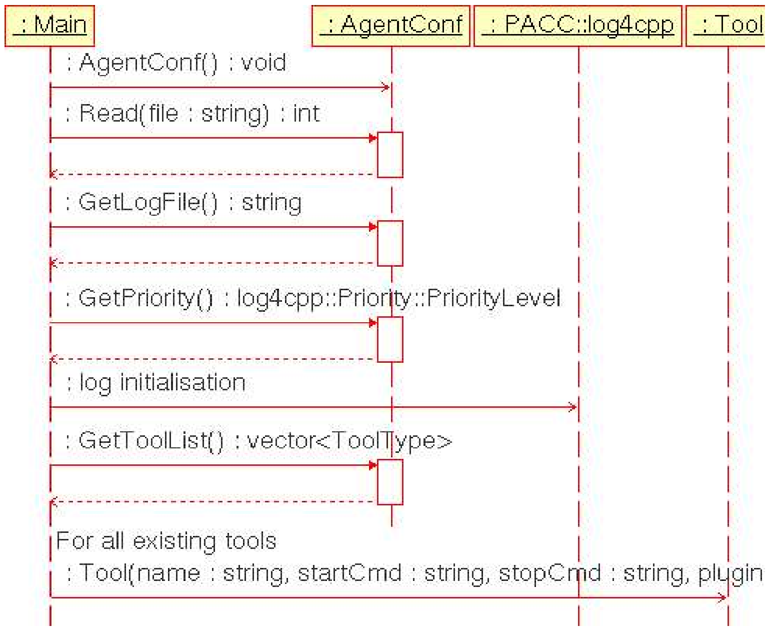
3.3.2 Configuration file

The configuration file (XML format) will contain the following informations :

- The manager IP and port.
- The agent server listening port.
- The log file name and a log priority.
- The list of tools installed on the host with an unique name, the start command, the

stop command, the plugin name, and some option (to said if the tool should be run at the agent start-up for example).

3.3.3 Sequence diagrams



: Main
: AgentConf
: PACC::log4cpp
: Tool
: AgentConf() : void
: Read(file : string) : int
: GetLogFile() : string
: GetPriority() : log4cpp::Priority::PriorityLevel
: log initialisation
: GetToolList() : vector<ToolType>
For all existing tools
: Tool(name : string, startCmd : string, stopCmd : string, plugin

Illustration 3Initialisation of the agent

This is the initialisation sequence. We first read the configuration file. We are initialising the log class after because it is in the configuration file that we can find the name and path of the log file and the log priority. From the configuration file we can create a list of the installed tool. This is acceding by this list that we will perform action on the tools.

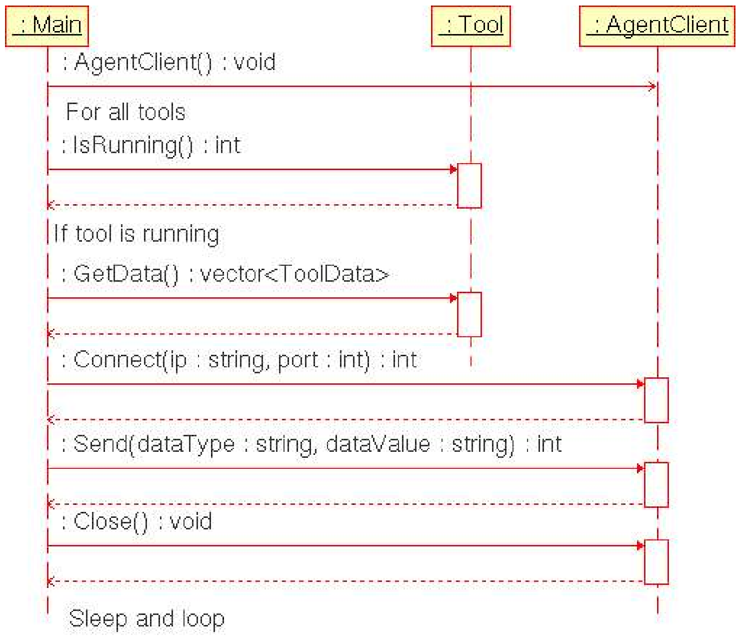


Illustration 4Tool reader thread

This is the thread who will regularly read the tools informations and eventually send it to the manager. We are getting data of tool only if the tool is launched. If there is data we open a connection with the manager and we send it.

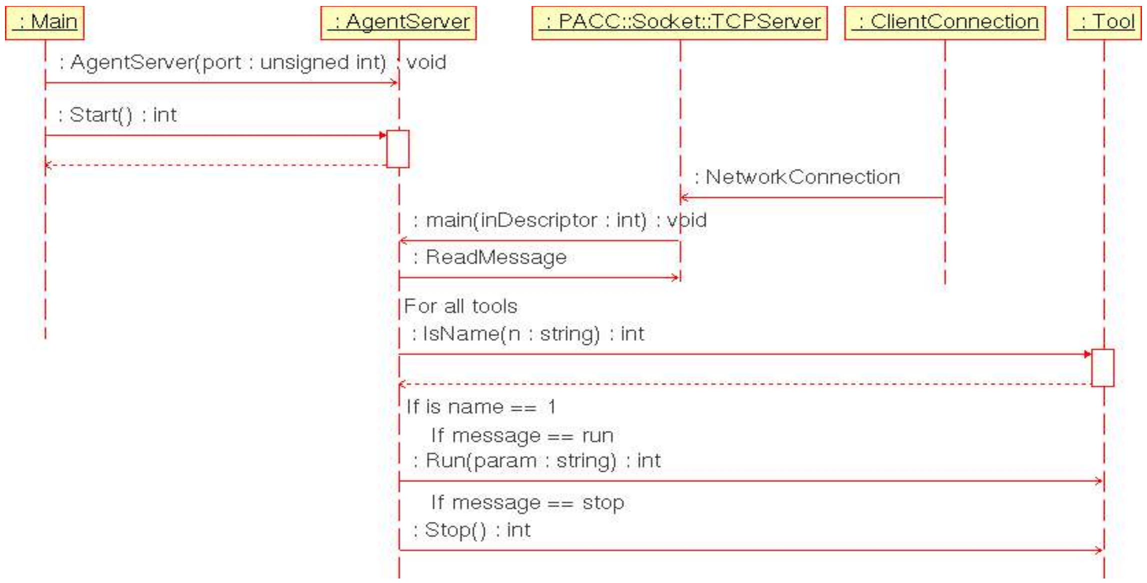


Illustration 5Server thread

This is the thread which is waiting for network connection from the manager. The main function (global) is only in charge of creating and starting the server. When a network connection occur, the main function of the AgentServer class (inherited from PACC::Socket) is executed. This function read the message, parse it for getting informations and performing the action of the message. First it has to find the tool corresponding with the tool named in the message. Then it will start or stop the tool, depending one the order.

3.3.4 Class diagram

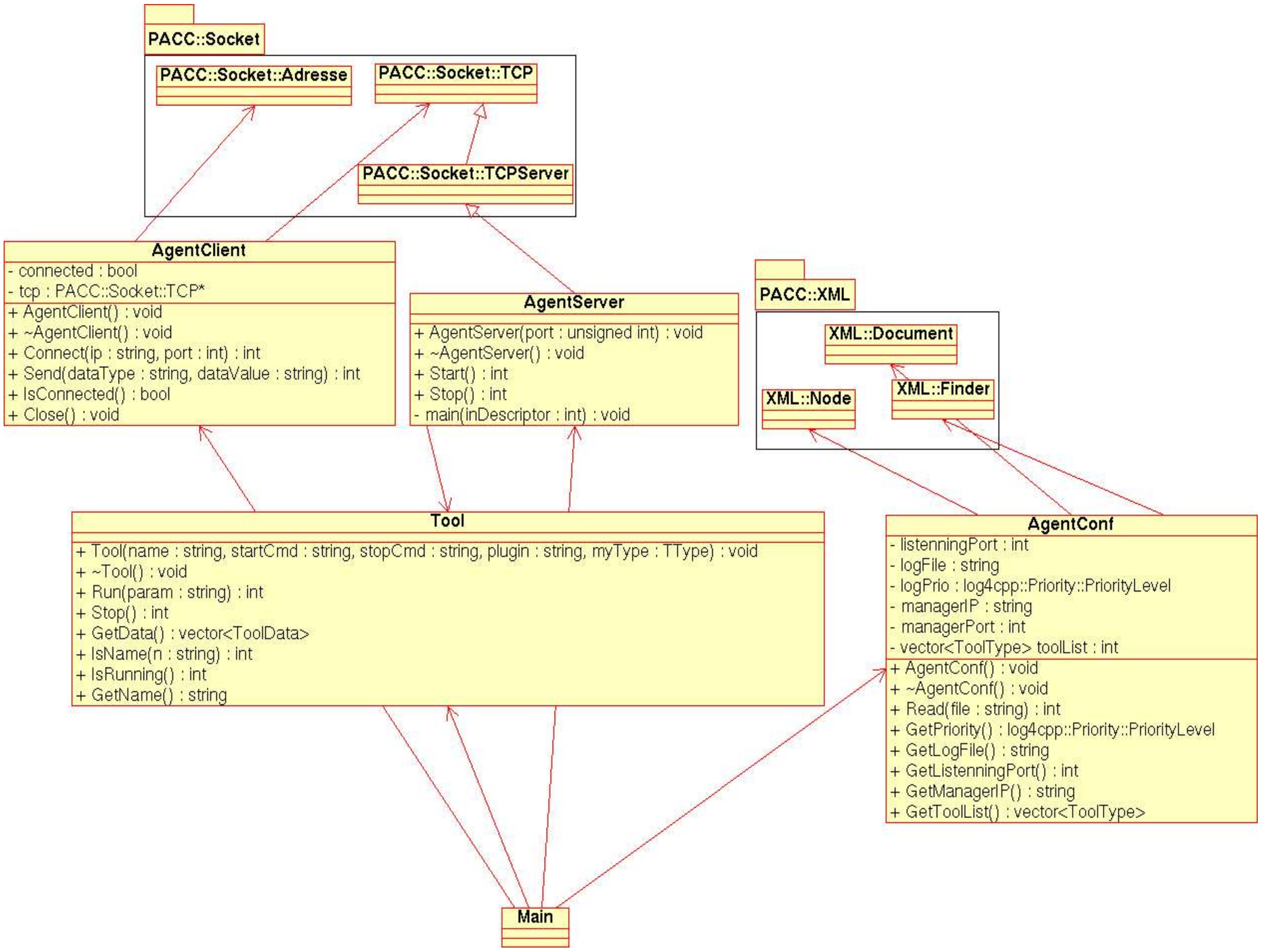

PACC::Socket
PACC::Socket::Adresse
PACC::Socket::TCP
PACC::Socket::TCPServer
AgentClient
- connected : bool
- tcp : PACC::Socket::TCP*
+ AgentClient() : void
+ ~AgentClient() : void
+ Connect(ip : string, port : int) : int
+ Send(dataType : string, dataValue : string) : int
+ IsConnected() : bool
+ Close() : void
AgentServer
+ AgentServer(port : unsigned int) : void
+ ~AgentServer() : void
+ Start() : int
+ Stop() : int
- main(inDescriptor : int) : void
PACC::XML
XML::Document
XML::Node
XML::Finder
Tool
+ Tool(name : string, startCmd : string, stopCmd : string, plugin : string, myType : TType) : void
+ ~Tool() : void
+ Run(param : string) : int
+ Stop() : int
+ GetData() : vector<ToolData>
+ IsName(n : string) : int
+ IsRunning() : int
+ GetName() : string
AgentConf
- listenningPort : int
- logFile : string
- logPrio : log4cpp::Priority::PriorityLevel
- managerIP : string
- managerPort : int
- vector<ToolType> toolList : int
+ AgentConf() : void
+ ~AgentConf() : void
+ Read(file : string) : int
+ GetPriority() : log4cpp::Priority::PriorityLevel
+ GetLogFile() : string
+ GetListenningPort() : int
+ GetManagerIP() : string
+ GetToolList() : vector<ToolType>
Main

Illustration 6Agent class diagram

A part of the structure of the software depend of the existing classes we are using, like PACC::XML, PACC::Socket::TCP and PACC::Socket::TCPServer.

To have a software easy to evolute we need to chose the class such have we can re-write some important functionalities, without changing the interface and the other classes.

The functionalities of the agent are :

- The network communication server side, agentServer : The class who will be in charge of it must inherit the PACC::Socket::TCPServer. This is the main function who will perform some action when there is a client connection. The overload of the main function is compulsory and we need this class. Moreover, it allow to define an interface who won' tchange if we wont to change the network communication classes (PACC::Socket).
- The network communication client side, agentClient : This class is in charge to manage to connect to a server and to send a message to it. It will implement all the communication protocol and present a simple and generic interface. Whit it, the software is independent of the implementation of the network communication.
- The configuration management, agentConf : Like the other class, this class is here to present an interface which is independent on the way we are implemented the configuration parser.
- The Tool : This is the only true class has they will be more than one instance of it. Each instance represent a Tool and ones can perform action on it, like reading data, starting or stopping.

3.4 Manager

3.4.1 Functionalities

The Manager is the art of the system. It is receiving data from the Agents, performing the decision process (Intelligent part) and sending order to agents, depending on the decision.

- **Receiving data from the agents** : The Manager act as a server, waiting for Agents connection. We can have many connections in the same time and we need to be able to manage it. The message received by the manager is detail in the communication protocol. After receiving those data, we need to send it to the decision process.
- **Decision process** : The is the intelligent part of the system. From inputs (tools data, rules and security policy) it will generate an output who can be an order (stopping/launching a tool) and/or an alerting.
- **The rules** : The rules must be defined by the user. It describe the decision to take, depending on the tools data. The goal is to make a decision tree. This tree will define the different step of a scenario, to react on an alert. Each time we receive new data, we will be able to go further in the tree. For writing rules, the user will need to know exactly which data the tools can send and which tool is on which host (to write the decision). We will provide functions to get the data and to set the decision. The ideal is to define a language using XML, who will allow to write comprehensible rules, by human and computer. It will also allow verification in the form, and moreover in the logic (using some expert system like prolog [Tool5]).
- **The security policy** : This allow to put some priority on some objects (data). The object will be the user/User group, the IP/Network and the time of the day. This security policy can be used in the rules, to make different decision depending of the priority of the object. For example, we won't take the same decision if the user is a student in an hostel or an administrator in a laboratory. The same, we can be tolerant if a student is using lot of bandwidth during the night. We will need some function in the rules language to use this security policy. We will store this security policy in a database and use a web interface to tune it. The policy used in the OSSIM [Tool2] project is a good example.
- **Alerting** : We also need to alert the user. The alert process can be quite sophisticate but at this stage we will only report it in a database that can be consulate by a web interface.
- **Sending the order to the agents** : when we have a decision, we need to send it to the appropriate agent. For this we need the list of the agent with there IP and port. We, then, will connect to the agent and send the order. The message is detailed in communication protocol.

3.4.2 Implementation choices

Looking at the required functionalities, we can figure out that we will need different thread : A set of thread (server threads) for the server. We would have to treat more than one incoming connection at the same time. Each connection will generate a different thread. The other thread is the decision thread. It will receive data from the server threads, perform the decision process and send the orders to the agents.

The server threads need to send the received data to the decision thread. We will use a message queue (POSIX object).

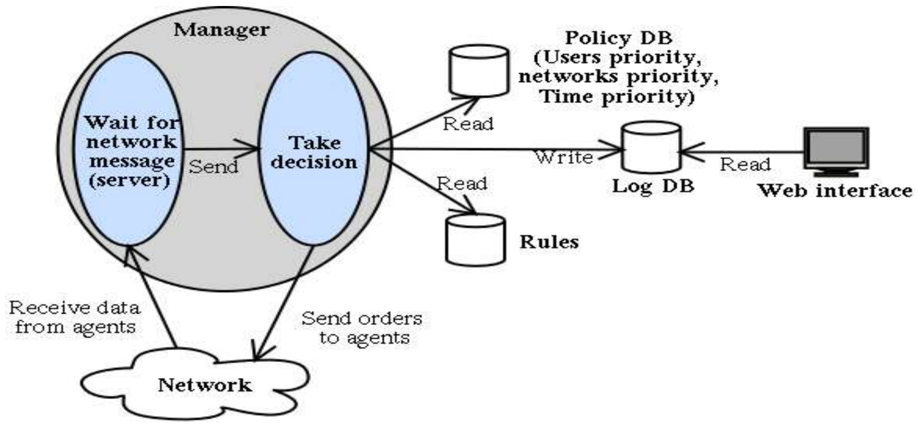


Illustration 7Manager' soperations

We will use the same library and class than for the Agents. The server and client part will be quite similar.

For the database (alerting and security policy) we will use a MySQL database. The C++ interface to use MySQL is given on the MySQL web site. For the user interface (tuning the security policy and consulting alerts) we will use PHP/HTML

3.4.3 Configuration file

The configuration file (XML format) will contain the following informations :

- The Manager server listening port.
- The log file name and a log priority.
- The list of Agents with there name, IP and port.

3.4.4 Sequence diagrams

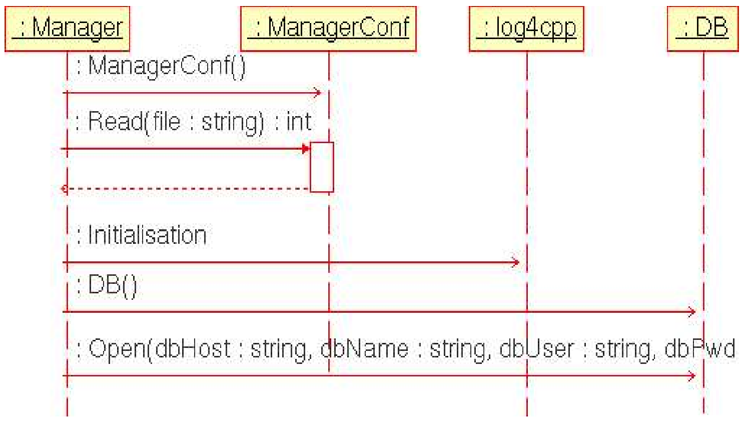

: Manager
: ManagerConf
: log4cpp
: DB
: ManagerConf()
: Read(file : string) : int
: Initialisation
: DB()
: Open(dbHost : string, dbName : string, dbUser : string, dbPwd

Illustration 8Manager initialisation

The initialisation of the manager is composed of the reading of the configuration file, the initialisation of the log and the initialisation of the database.

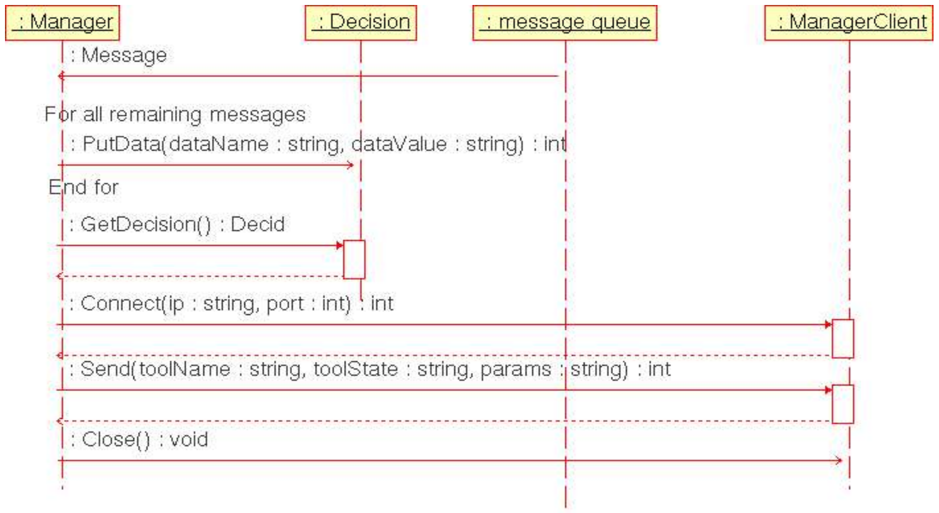

: Manager
: Decision
: message queue
: ManagerClient
: Message
For all remaining messages
: PutData(dataName : string, dataValue : string) : int
End for
: GetDecision() : Decid
: Connect(ip : string, port : int) : int
: Send(toolName : string, toolState : string, params : string) : int
: Close() : void

Illustration 9Manager decision thread

The decision thread is activated from received message from the message queue. For all new message it will put it in the Decision class. When all information is put, it will Call GetDecision. This function will perform the decision process by reading the decision tree (the rules) and try to take a decision.

If there is a decision, we are connecting to the appropriate Agent and sending the decision.

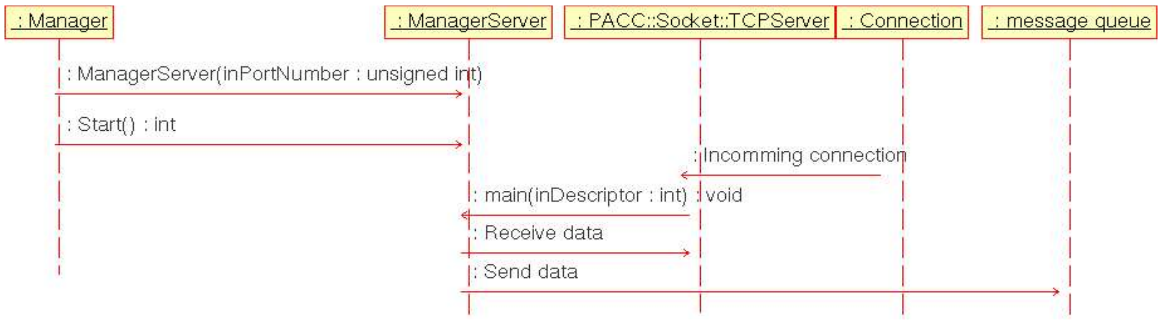


Illustration 10Manager server thread

The server thread of the manager act like the one of the Agents. It is waiting for a agent connection. The main function of ManagerServer will then receive the data, parse it and send it in the message queue for the decision thread.

3.4.5 Class diagram

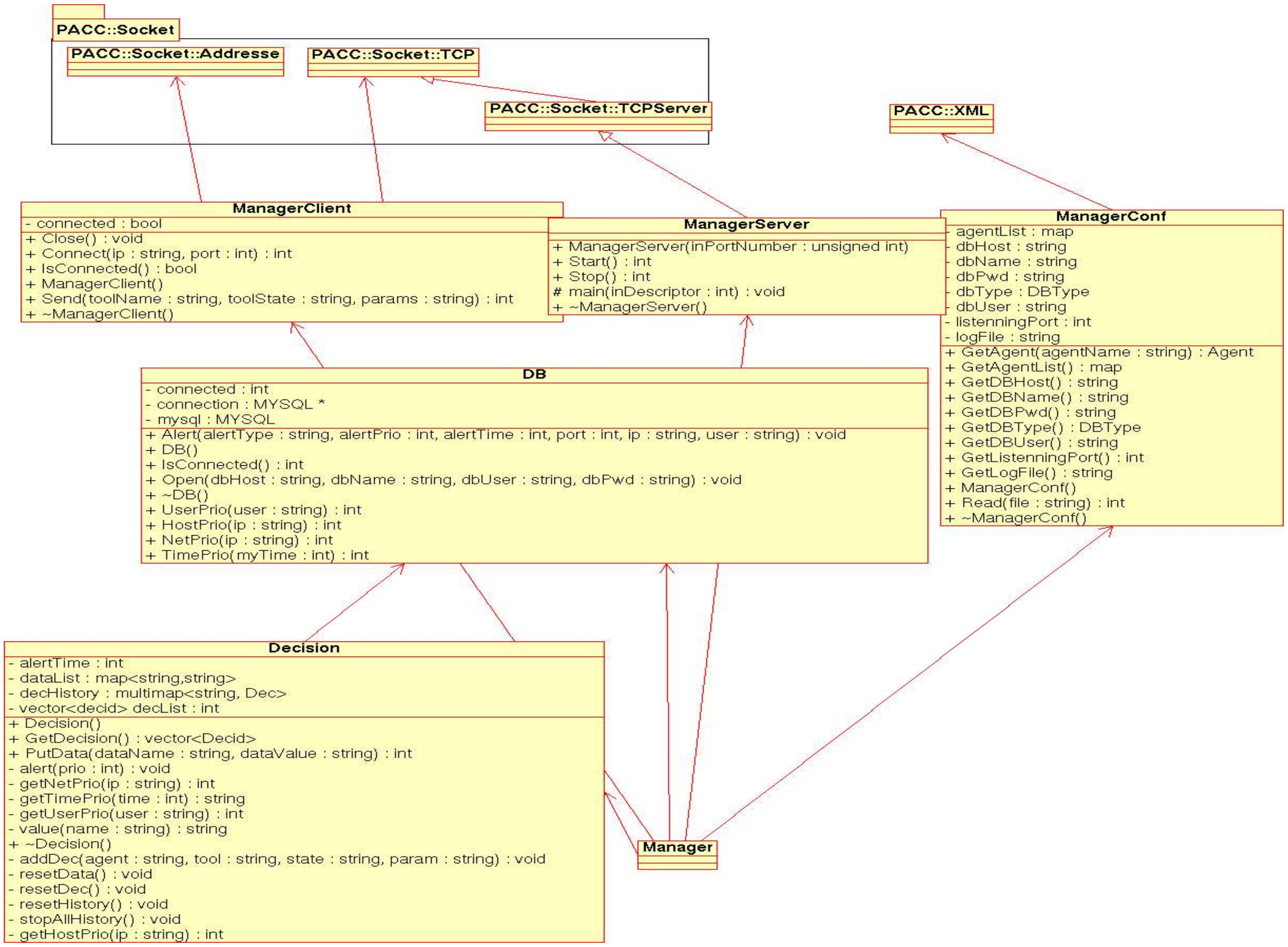

PACC::Socket
PACC::Socket::Addresse
PACC::Socket::TCP
PACC::Socket::TCPServer
PACC::XML
ManagerClient
- connected : bool
+ Close() : void
+ Connect(ip : string, port : int) : int
+ IsConnected() : bool
+ ManagerClient()
+ Send(toolName : string, toolState : string, params : string) : int
+ ~ManagerClient()
ManagerServer
+ ManagerServer(inPortNumber : unsigned int)
+ Start() : int
+ Stop() : int
# main(inDescriptor : int) : void
+ ~ManagerServer()
ManagerConf
- agentList : map
- dbHost : string
- dbName : string
- dbPwd : string
- dbType : DBType
- dbUser : string
- listenningPort : int
- logFile : string
+ GetAgent(agentName : string) : Agent
+ GetAgentList() : map
+ GetDBHost() : string
+ GetDBName() : string
+ GetDBPwd() : string
+ GetDBType() : DBType
+ GetDBUser() : string
+ GetListenningPort() : int
+ GetLogFile() : string
+ ManagerConf()
+ Read(file : string) : int
+ ~ManagerConf()
DB
- connected : int
- connection : MYSQL *
- mysql : MYSQL
+ Alert(alertType : string, alertPrio : int, alertTime : int, port : int, ip : string, user : string) : void
+ DB()
+ IsConnected() : int
+ Open(dbHost : string, dbName : string, dbUser : string, dbPwd : string) : void
+ ~DB()
+ UserPrio(user : string) : int
+ HostPrio(ip : string) : int
+ NetPrio(ip : string) : int
+ TimePrio(myTime : int) : int
Decision
- alertTime : int
- dataList : map<string,string>
- decHistory : multimap<string, Dec>
- vector<decid> decList : int
+ Decision()
+ GetDecision() : vector<Decid>
+ PutData(dataName : string, dataValue : string) : int
- alert(prio : int) : void
- getNetPrio(ip : string) : int
- getTimePrio(time : int) : string
- getUserPrio(user : string) : int
- value(name : string) : string
+ ~Decision()
- addDec(agent : string, tool : string, state : string, param : string) : void
- resetData() : void
- resetDec() : void
- resetHistory() : void
- stopAllHistory() : void
- getHostPrio(ip : string) : int
Manager

Illustration 11Manager class diagram

The role of the ManagerServer, ManagerClient and ManagerConf are the same as the agent, I won't repeat it.

The DB class also play the role of an interface to hide the implementation of the database. It offer hight level function to accede and add information to the database.

The Decision class is the art of the Manager. It is providing both public functions to add informations in the decision process and to get the decisions. It also provide private function which are used in the rules. Actually those functions are necessary because the rules are only a C file which is include in the Decision class. In the future when the rules will be in XML, those functions will disappear but we will have to add an XML parser.

3.5 Communication Protocol

We are using the TCP protocol. We will send string messages. Each message separate by a line break (\n).

We will always send a time stamp with the messages.

3.5.1 Message sending by the Agents to the Manager

We will first send the agent name : (name is the name of the agent in the configuration file but AgentName is a constant string)

```
Time:AgentName:name\n
```

And after the tools data : (DataName are names defined in the plugins)

```
 Time:Data1Name:Data1Value\n
 Time:Data2Name:Data2Value\n
```

3.5.2 Message sending by the Manager to the Agents

```
Time:ToolName:ToolState:Params\n
```

ToolName is the name of the concerning tool on the agent, ToolState can be run or stop and params are parameters for running the agents, it is usually empty (“”) for stopping.

3.6 Prototype limitations

For this first prototype we will put some limitations :

- The main limitation is that the rules are hard-coded. When then need to recompile the manager each time we are changing it. Moreover, the rule language is just the C++ language using some functions to get data, setting the decision and alerting.
- No verification are performed on the rules.

- We won't take into account more than one alarm at the same time. Indeed there is only one decision thread running, who will treat one decision process. If an other alarm come during the treatment of an other alarm, we will ignore it.
- The security policy only take user into account and no user group. Getting user group from an user will need interaction with LDAP or Unix user system (for example).
- The communication protocol as no security (like SSH).
- The user interface is very simple.

4 TOOLS

Using tools are presented in the [Doc1]. But we will give some more details here.

4.1 Hight level anomaly detection

The entry point of our system is an hight level data collection. Those collected data can be bandwidth (on main router/firewall/switch), number of connections on a proxy server, number of mail sent ... This data collection should not use a lot of resources and we want to be able to detect anomalies.

Those data are time dependant variables. The difficulties in detection abnormal behaviour is in the fact that those data are moving, mainly depending on the time of the day (but also on the day in the week and on the month). So we can' tuse a fixed threshold to raise alarms, we must have some learning of the network behaviour.

I was exploring two solution to perform network behaviour learning and anomaly detection :

- The first method is using a learning tree algorithm. One can construct a tree from a set of data. Those data must first be collect and fill by an operator. After the tree is constructing, new data can be automatically classified. The operator can, then, correct the decision if it is wrong and launch the learning algorithm again. We can assume that after a will, little errors should be made.

  In our case, we can collect information when it exceed an certain threshold. Information can be the date in the month, the day of the week, the month, the bandwidth (or number of proxy connection...). From those information and with an operator help, we can construct the tree to find cyclic bandwidth excess.

  The main problem of this technique his the learning part. It can take a long time and need many work of the operator. It is impossible to determine how many time it can take.

  The main interest his, when it his well tune, we can assume a very little false positive alarms.

One of the more interesting algorithm to perform this method is [Doc2]. It provide construction of tree for discrete and non discrete data. A set of tools that implement it can be found on C4.5 Tutorial :
http://www2.cs.uregina.ca/~hamilton/courses/831/notes/ml/dtrees/c4.5/tutorial.html

- The second method is using Holt-Winter Forecasting Algorithm. This algorithm is describe in [Doc3]. It try to predict future values from older existing values. It is based on the premise that a time series can be decomposed into three components: a baseline, a linear trend, and a seasonal effect. The algorithm presumes that each of these components evolves over time. Then, after a time of ' learning', the algorithm is able to detect values which not fit with the predicted interval.

  The algorithm can be tune with some adaptation parameters. We can choose if prediction reflect recent observation of time series or older ones. There is also a system of window and threshold. To have en alarm, the time series must exceed the predicted value more than a define number of time within a define window.

  Actually, these algorithm is implemented for Rond Robin Databases [Tool6]. It is implemented in the development version (1.1.0) of RRDTools (Application for using RRD).

  Every software using RRDTool can use this algorithm with no ore little modifications.

  Bandwidth graph generated with MRTG using aberrant behaviour detection. The blue line denote the bandwidth, the red lines form the confidence bands and the yellow lines denote the alerts when the bandwidth goes out of the confidence bands :

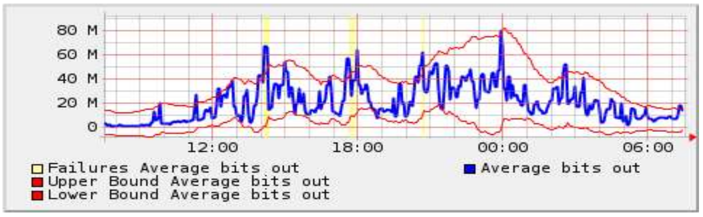


Illustration 12 : Bandwidth graph generated with MRTG using aberrant behaviour detection. The blue line denote the bandwidth, the red lines form the confidence bands and the yellow lines denote the alerts when the bandwidth goes out of the confidence bands.

  The advantage of this method his that it doesn't need any operator intervention to learn the network behaviour.
  In the other side, it is not able to detect cyclic events on long time (more than one day) and can produce some false alarms.

We choose the Holt-Winter Forecasting method. The C4.5 method take too much time and need to much work to learn the network Behaviour. Moreover, the chosen method need little development has it is used in others projects, like Cricket [Tool1] or OSSIM.

To perform the data collection, we chose to use MRTG. This tool is already used on the actual network, easy to configure, and correspond to our requirement.

First, it is compatible with the Aberrant Behaviour Detection, has it can log data in RRD using RRDTools. The only problem is the Database creation. At the first start, if no database exists, MRTG will create it. But it is not envisaged for giving the necessary parameters for anomaly detection. This is easy to solve by creating manually the database, or using the patch providing in the OSSIM project who add this functionalities.

MRTG, then, allow us to perform data collection with SNMP variables, which don't cost a lot of resources. If no SNMP variable is available, we can even use some scripts to collect data from logs, for example, and to send it to MRTG. This process is most costly has it require log analysis.

4.2 Log analysis

In the aim of optimising the log analysis, we suggest to use Multilog [Tool8]. This log system can be use with several tools. It has the particularity to separate logs in small files. Each file as a name with a time stamp which represent the last time stamp of the file.

Then, to look for some information at a given time, we will first have to find the good file and then the good line in the file. This is quicker than parsing an big log file from the beginning.

If we can't use Multilog, we also think of a method who will remember the last position we where in the fill, and start to parse it from her.

Actually we are providing some log analysis with there plugins :

- A squid log analyser who will figure out the user name and IP using the biggest bandwidth in a given interval of time.
- A squid log analyser who will figure out the user name and IP who make the most number of connection to the proxy in a given interval of time (to detect virus for example).
- A Qmail log analyser to figure out the user name and IP sending the most mail in a given interval of time.

5 Documentation

5.1 Installing the framework

5.1.1 Installing the Agents

On each host where there is tools we want to use, we need to install an Agent.

The installation require :

- The log4cpp library and source (for compilation). You can download it on log4cpp.sourceforge.net

1. Uncompress the INMF tar.gz file : `tar -xvzf  INMF-0.1.tar.gz`
2. Go in the INMF/agent directory
3. In the agent directory : `./configure`
4. If there is no error, you can compile the agent : `make`
5. Install the agent : `make install`
6. Create a directory `/etc/INMF`
7. Copy the config file `INMF/contrib/agent.cfg` in this directory

You can find more detailed informations in the INSTALL file.

5.1.2 Configuration of the Agents

The default configuration file is in /etc/inmf/agent.cfg

This file is commented to allow you to make your own configuration.

5.1.3 Installing the Manager

On one host who can accede the entire managed network, you need to install the manager.

The installation require :

- The log4cpp library and source (for compilation). You can download it on log4cpp.sourceforge.net
- MySQL server
- The MySQL interface for C++, you can download it on dev.mysql.com
- An apache server with PHP

1. Uncompress the INMF tar.gz file : `tar -xvzf  INMF-0.1.tar.gz`
2. Go in the INMF/manager directory
3. Change the rule file : `rules.c`
4. In the manager directory : `./configure`
5. If there is no error, you can compile the manger : `make`
6. Install the manager : `make install`
7. Create a directory `/etc/INMF`
8. Copy the config file `INMF/contrib/manager.cfg` in this directory

You can find more detailed informations in the INSTALL file.

5.1.4 Configuration of the Manager

The default configuration file is in /etc/inmf/manager.cfg

This file is commented to allow you to make your own configuration.

The rule file is in src/rules.c

To write the rules you need to have exactly :

- The name of the agent as define in the configuration file. This is to avoid using directly the agent IP in the rules.
- The installed tools on each agent, and there name (referring to the Agents config file).
- The name of the initial alert you can get (see plugins documentation).
- The name and the signification of the data each agent can send (see plugins specification).

The rules language is actually C++. Some functions are provided :

- string value(string DataName) : Return the corresponding value of the data with name DataName. Return a empty string if the data doesn't exist.
- alert(int priority) : Send an alert to the user. Actually write this alert in the MySQL database. The priority is between 1 (low) and 10 (hight).
- int getUserPrio(string user name) : Look in the security policy database the priority affected to this user. The priority is between 1 and 5. If the user is not in the database, the priority will be 0 by default.
- int getNetworkPrio(string IP) : Look in the security policy database the priority affected to the network containing this IP. The priority is between 1 and 5. If the IP doesn't belong to any network in the database, the priority will be 0 by default.
- int getIPPrio(string IP) : Look in the security policy database the priority affected to

this IP. The priority is between 1 and 5. If the IP have no entry in the database, the priority will be 0 by default.

- int getTimePrio(string myTime) : Look in the security policy database the priority affected to the Time. The priority is between 1 and 5. Doesn' t belong to any time interval in the database, the priority will be 0 by default.
- logger.info(string text) : To write informations in the log file.

To define the decision we have to give value to :

- addDec(string agent, sting tool, string state, string params) :
  - agent : The name of the concerned Agent.
  - tool : The name of the concerned tool on the Agent.
  - state : “run” or “stop” for running or stopping the tool.
  - params : The parameters we want to give to the command for running the tool. Usually empty for stopping tool.
- resetHistory() : reset the list containing all the decision take. We have to use it when we finish one decision process to state a new decision process.
- stopAllHistory : stop all the previously launched tool. We have to use it when we finish one decision process to state a new decision process.

Using this we can construct a decision tree, mainly using the C++ if(condition) {} function. The files in the distribution can be use as an example.

5.1.5 Installation of the web interface

The installation require :

- A HTTP server (like apache) with php.
- It must be install on a server who can accede to the MySQL database of the Manager.

1. Uncompress the INMF tar.gz file : `tar -xvzf INMF-0.1.tar.gz`
2. Put all the files of the `INMF/web` in a directory where the HTTP server can accede it.
3. Configure your HTTP server.
4. Edit the `db.php` file and set the MySQL server address, the user and password and the database name, in the start of the file.

5.2 Installation of the tools

5.2.1 MRTG (Multi Router Traffic Grapher)

We use the latest version of MRTG for monitoring various hosts. At the time of writing this document, the latest stable version is 2.10.14. A patch is required to enable MRTG for creating the RRD archives compatible with Aberrant Behaviour Detection (Using RRDTool 1.1.0).

The patch is available in the tools_proto0.tar.gz file, in the contrib/mrtg.diff file. Think you to the OSSIM project who make this patch.

- Download the MRTG sources at : people.ee.ethz.ch/~oetiker/webtools/mrtg/pub/
- Decompress it.
- Go in the mrtg/bin directory.
- Enter patch -p0 < pathToPatchFile/mrtg.diff
- Follow the normal mrtg installation guide.

You also have to install the development version of RRDTool (1.1.0). You can find it on : people.ee.ethz.ch/~oetiker/webtools/rrdtool

Here are examples on how to configure MRTG :

- Monitoring bandwidth: MRTG monitors the "ifInOctets" and "ifOutOctets" using SNMP requests. The configuration file for doing so is provided.

```
WorkDir: /path/to/the/directory
Options[_]: growright
RunAsDaemon: Yes
Interval: 5 #it means 5 minutes here , can be changed
LogFormat: rrdtool
PathAdd: /path/to/rrdtool/bin/
LibAdd: /path/to/rrdtol/lib/perl/
Target[Statistics]: interface:community string@host
MaxBytes[Statistics]: 12500000
```

Of course, one can add to configuration file for drawing graphs with mrtg. Here the data is directed to RRD database. Graphs can be drawn using RRDtool's graph function.

- Monitoring Squid : MRTG monitors various Squid SNMP variables. Out of them all, cacheServerRequests determine the number of requests.

```
WorkDir: /path/to/the/directory
Options[_]: growright
RunAsDaemon: Yes
Interval: 5 #it means 5 minutes here , can be changed
LogFormat: rrdtool
PathAdd: /path/to/rrdtool/bin/
LibAdd: /path/to/rrdtol/lib/perl/
```

```
            Target[cacheServerRequests]:
cacheServerRequests&cacheServerRequests:community@host:3401
            MaxBytes[cacheServerRequests]: 10000000
            Options[cacheServerRequests]: nopercent
```

5.2.2 Qmailmrtg

Qmailmrtg allow us to monitor QMail logs using MRTG. Using it, we can monitor the number of sending mails.

Qmailmrtg require the installation of Multilog (part of the daemontool package : http://cr.yp.to/daemontools.html)

Install qmailmrtg (http://www.inter7.com/index.php?page=qmailmrtg7) using the documentation.

In the configuration file , the following lines need to be added to enable RRD storage.

```
LogFormat: rrdtool
PathAdd: /path/to/rrdtool/bin/
LibAdd: /path/to/rrdtol/lib/perl/
```

5.2.3 TCPTrack

The original TCPTrack software is only showing data on screen. We have to read those data. For this, we have to modify the original software. The next version of TCPTrack will include functions to dump data in one file. Our modified version will run for 30 seconds and print on the standard output the data.

The patch is on the INMF/contrib directory of the project.

5.2.4 Scripts

We are providing several scripts to get informations from log files :

- squiduser.pl : This script is a parser for squid (proxy server) logs. It take an IP and an interval of time (StartTime, StopTime) in parameter and will find the login of the user using this IP in this interval of time. It will store the output in a file called squiduserlog in the format : `User:userid`

  The associated plugin for getting information from this script is plugin_squiduser.pl

  The path to the squid log file has to be set at the beginning of the file.

  This script is used in the abnormal bandwidth exceed scenario. When we have the IP and the port responsible of this exceed, we can get is login name on the proxy server, if the port correspond to a traffic who go thought the proxy server.

- squidcnx.pl : This script is a parser for squid (proxy server) logs. It take an interval of time (StartTime, StopTime) in parameter and will find the login and IP of the user making the maximum number of connection to the server. It will store the output in a file called squidcnxlog in the format : `User:userid\n IP:ip\n Usage:num of`

```
cnx\n
```

The associated plugin for getting information from this script is plugin_squidcnx.pl

The path to the squid log file has to be set at the beginning of the file.

This script is used in the abnormal proxy connection scenario. When know that there is an abnormal number of connection on the proxy server, we want to find who is responsible of it.

5.2.5 Plugins

- plugin_squiduser.pl : This script will read the dump file of the squiduser script, get the informations (user login) and reset the file. The return is :

```
User:login\n
```

- plugin_squidcnx_rrd.pl : This plugin is collecting data from RRD with aberrant behaviour detection. It is looking for abnormal number of connexion to the squid proxy server sent alarms in the database. If it detect an alarm, it return :

```
Alert:squidcnx\n
FirstFailureTime:Time\n
LastFailureTime:Time\n
```

- plugin_squidcnx.pl : This plugin will read the dump file of the squidcnx script, get the informations (user login and IP) and reset the file. The return is :

```
User:login\n
IP:ip\n
Usage:number of connexion\n
```

- plugin_bandwidth_rrd.pl : This plugin is collecting data from RRD with aberrant behaviour detection. It is looking for bandwidth exceed alarms in the database. If it detect an alarm, it return :

```
Alert:InBw\n if the alert is on the In bandwidth
Alert:OutBw\n if the alert is on the Out bandwidth
FirstFailureTime:Time\n
LastFailureTime:Time\n
```

- plugin_qmailsent_rrd.pl : This plugin is collecting data from RRD with aberrant behaviour detection. It is looking for abnormal number of mail sent alarms in the database. If it detect an alarm, it return :

```
Alert:qmailsent\n
FirstFailureTime:Time\n
LastFailureTime:Time\n
```

- plugin_tcptrack.pl : This plugin is analysing TCPTrack dump file and file get the IP and the port which is using the biggest bandwidth. It return :

```
IP:ip\n
Port:port\n
```

6 References

6.1 Web sites

[Tool1] Cricket : Tool for monitoring time series data, cricket.sourceforge.net

[Tool2] OSSIM : Open Source Security Information Management, www.ossim.net

[Tool3] PACC : Set of class for C++. We are using the XML and Socket classes, manitou.gel.ulaval.ca/~parizeau/doc/namespaces.html

[Tool4] log4cpp : Library to manage logs (file, screen ...) for C++, log4cpp.sourceforge.net

[Tool5] prolog : Logic programming language. A lot of web site but you can see, cs.wwc.edu/~cs_dept/KU/PR/Prolog.html

[Tool6] RRD : Rond Robin Database. Time series oriented database, people.ee.ethz.ch/~oetiker/webtools/rrdtool

[Tool7] MRTG : Multi Router Traffic Grapher, people.ee.ethz.ch/~oetiker/webtools/mrtg

[Tool8] Multilog, log tool. cr.yp.to/daemontools/multilog.html

6.2 Documents

[Doc1] Intelligent real-time reactive Network Management. Guillaume Andreys, Abhishek Jain and G. Sivakumar.

[Doc2] C4.5 : Programs for machine learning. J. Ross Quinlan. Morgan Kaufmann edition. 1993.

[Doc3] Aberrant Behaviour Detection in Time Series for Network Monitoring. Jake D. Brutlag. 2000. http://www.usenix.org/events/lisa2000/full_papers/brutlag/

[Doc4] Intrusion detection system. Bibliographic report. Guillaume Andreys.