UNIVERSITY “POLITECHNICA” OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTERS

Malicious clients’ detection in BlobSeer

Supervisors:

Prof.dr.ing. Cristea Valentin

As.drd.ing. Leordeanu Cătălin

Author:

Lepăr C. Ana-Maria

-2010-

Abstract

This thesis addresses the problem of malicious client detection in BlobSeer, a large-scale distributed storage system. Through the careful analysis of the user history we can detect the actions which fall outside the normal operation of the system. We can also determine a trust level for each user which represents the probability that a user is harming the storage system. We use this trust level to develop a complex system which rewards the fair users and restricts the activities of the malicious ones.

Keywords

Blob, malicious clients, distributed systems, trust level, security

Contents

1. Introduction 4
   1.1. Context 4
   1.2. Motivation 4
   1.3. Structure 4
2. State of the art 6
   2.1. Cloud computing – a new trend 6
   2.2. Security in large distributed systems 8
3. BlobSeer 11
   3.1. Main features 11
   3.2. What a client can do 13
   3.3. Security and malicious clients 14
4. Malicious clients detection module 17
   4.1. Theoretical issues 17
   4.2. Implementation 21
   4.3. Interaction with other BlobSeer modules. Input and output data format 28
5. Experimental results and conclusions 29
6. Future work 33
Annexes 36
   Annex 1: BlobSeer Installation 36
   Annex 2: BlobSeer configuration scripts 44
   Annex 3: BlobSeer operations 47
   Annex 4: MonALISA installation 48
   Annex 5: Database content – table policy_events: 49
   Annex 6: Input and output files examples 50

1. Introduction

1.1. Context

In our days is cheaper to borrow hardware in order to deploy software applications. In this context, cloud computing is not only a thing, but it describes a set of transformations that appeared in IT, during in the last 2 years.

In this context, KerData team from INRIA, France, developed a scientific system, called BlobSeer, a data storage service specifically designed to deal with the requirements of large-scale data-intensive distributed applications. In this project is also involved University Politechnica of Bucharest, the DSlab team.

BlobSeer is composed of many modules, and like any system, it needs a module of security.

1.2. Motivation

This thesis addresses how monitoring users' activity can be used to protect the system, by getting each user a set of rights.

The malicious clients’ detection module is created to be used in any distributed system, because it has a very big grade of abstractization. We used it in BlobSeer, where we compute the trust for each client, and using this trust, we classify them in good and malicious clients.

1.3. Structure

The thesis is composed of five chapters, where we describe the problem, the solution we designed, the system where we apply the solution, experimental results, conclusion and future work.

In the first chapter, we describe the context: cloud computing - the implication of this new trend in science and business and one of the major problem in this kind of systems, the security.

The second one describes the system where we implement the solution, BlobSeer. Here you can find the system architecture, the main features and the problems that we find in the system security.

The third one is dedicated to the solution description: the security module architecture, the classes, input data format, output data format, how the module interacts with other system components.

The fourth chapter shows our experimental results and our conclusions and the last one presents our future work in this direction.

At the end of the thesis, you will find the references used to design this solution and annexes with BlobSeer installation, configuration files.

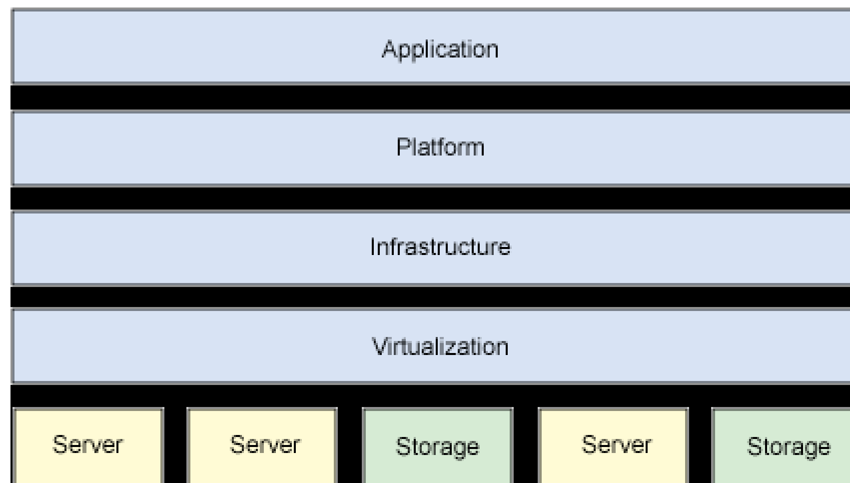
2. State of the art

2.1. Cloud computing – a new trend

Many companies and scientific communities promote the idea of renting virtual machines, storage space or already deployed software platforms, instead of buying and maintaining them. In this situation, a new concept has been developed, *cloud computing*.

The infrastructure of this kind of systems consists of services and data centers. There is a single access point for all costumers' needs, who don't own physical infrastructure. The architecture is composed of cloud components. These communicate each other over web services. Each component does only one thing well, like in Unix philosophy. In this case, we can say that the system is composed of monolithic pieces.

A cloud computing system is composed of five layers: *the client*, *the application*, *the platform*, *the infrastructure* and *the servers*. The client represents the computer hardware and software, used to deliver cloud services. Next layer is known also as SAAS (Software as a Service), because the software is delivered over the Internet and in this case, the user doesn't need to install applications on his own computer. The third layer is also known as PAAS (Platform as a Service). Its role is to deliver a computing platform and a solution stack for the service. It consumes the cloud infrastructure and sustains cloud applications. An important role is to facilitate the applications' deployment without a big cost of complexity. Next layer, IAAS (Infrastructure as a Service), delivers computer infrastructure, used by the applications. In fact, it is a platform virtualization environment seen as a service. The last layer, which is the basic one, consists of servers and computer hardware, necessary to deliver the services for the upper layers. In the image below, you can see the five layers and the interaction between them:



Cloud computing – Interaction between the five layers

Analyzing its architecture and structure, we can say that cloud computing comes to focus on increasing capacity, add capabilities on the fly, without investing in new infrastructure and training new personnel or licensing new software.

Like all new technologies, it has also strong and week points. In large scale distributed systems, such as Clouds, security is a major issue. This is a difficult challenge, because in this kind of systems, we have large numbers of clients from geographical regions. We also have to deal with large amounts of traffic and a high density of requests, so we need to take into account the performance of the system when designing the security layer.

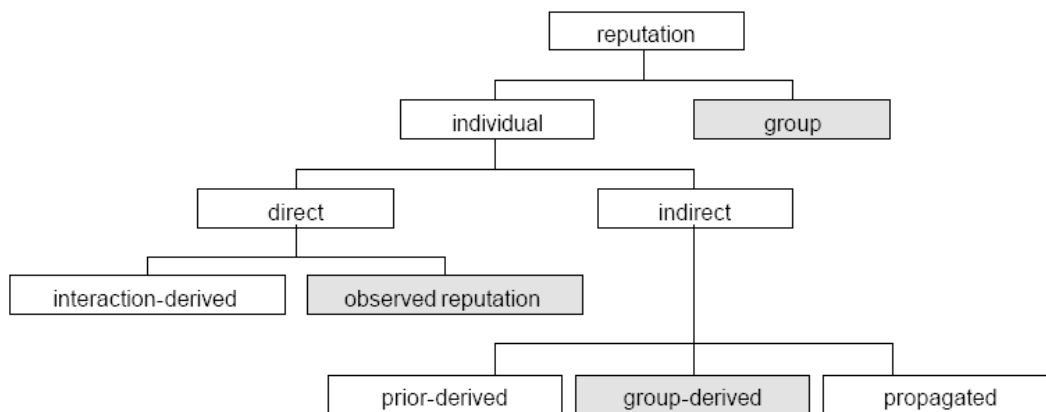
Despite this problem, cloud computing is used in projects developed by many universities and companies, like Carnegie Mellon, MIT, Indiana University, University of Massachusetts, University of Maryland, IBM, Oracle.

BlobSeer can be used successfully in a Cloud environment. It is used to store large and unstructured data blocks, called BLOBs. Each user may create, write and read this BLOBs. This makes BlobSeer a very efficient data management solution for large scale distributed systems.

2.2. Security in large distributed systems

A distributed system is not only a collection of hardware and software, but, also, a large virtual community. These kinds of communities have three advantages: being vast, nearly anonymous and simple to join or leave. When we have these features, we also think about security problems, because the members can be good or bad. In this case, we introduce a new concept of *the trust*, which is related to *reputation concept*.

Reputation is defined as a perception that an agent has of another’s intentions and norms. We find this term not only in computer science, but also in biology, economy and other fields. It is considered that reputation is a context-dependent concept. The scientists consider that there are many notions of reputation, like: individual reputation, group reputation, direct or indirect reputation. You can see bellow a reputation typology which was presented in [3]:



Reputation typology

In the top, there is individual and group reputation. There are systems based on individual reputation like: Amazon, eBay, Free Haven and others based on group

reputation: OpenRaitings or Reputation.com, where the group is studied from a firm perspective. In this case, the reputation is computed as an average of the group members’ reputation. The direct reputation is modeled using the experience with that user; meanwhile the indirect one is based on second-hand evidence. The direct reputation can be computed using the interaction with that user or from observation made. The indirect reputation is divided into three: prior-derived which is supposed that agents bring with them prior beliefs about strangers (in our case new users), group-derived where the client reputation is based on the group from where he takes part and finally, propagated reputation which consists on information garnered from others in the environment.

Nowadays, the specialists use a lot of modalities to compute and define the trust level for a user from a given community. They define the trusting interactions online in three ways: escrow services, history reporting for members and reputation rating for systems.

Escrow services provide the equivalent of institutional guarantees for virtual communities. They intent to ensure security by acting as a third party in transactions, where the two parties are not sure of each other's credibility. Rather than sending items directly to the other party, the first entity sends them to the escrow service, which sends the goods to the second party, if it is credible to it. If either party fails to deliver their part of the deal, the item sent to escrow service will be held and eventually returned to the party that sent it.

The history reporting for members is used in companies such eBay to provide a breakdown of the number of positive, neutral, and negative ratings, and written feedbacks for a given seller over 1 week, 1 month and 6 months prior to any given transaction. In this case, the information about users’ activity is stored in a special module, from where it is accessed when buyers are engaged in a transaction with a seller.

The “reputation rating” for systems provide succinct summaries of a user’s history or a given virtual community. The reputation is represented by a rating “score” which is

calculated based on cumulative (or average ratings) by its members. These systems where first implemented in e-commerce applications (Lik Mui, 2002).

Another keyword for security in distributed systems is “malicious client”. This can be defined as a user that broke one or more security policies. So, in this case an important component of a system is the module where are defined the security policies. We can see that the notion of “malicious client” is dependent to the notion of “security policy”, because we define this term using the last one.

The approach presented in this thesis uses these mechanisms to build a trust and reputation system which can be used to detect the users who don't respect the existing security policies or may be harming the system.

Malicious clients are found in many applications that are accessed via the Internet, so they represent a threat for system stability and functionality.

3. BlobSeer

3.1. Main features

Blobseer is developed by the KerData team, from France and DSlab, from UPB, Romania. The main function of this system is to store huge amounts of data and provide access to read/write/append requests from users. The environment is a concurrent P2P one.

BlobSeer provides checkpoints, roll back facilities (because are kept all the versions of a file in the system) and scalability.

The clients have to login and they can write or read data from the space reserved for them. They cannot delete or remove data, which is stored in versions on the file system.

The communication between the system entities is done using RPC. The processes are organized as a distributed hash table (DHT).

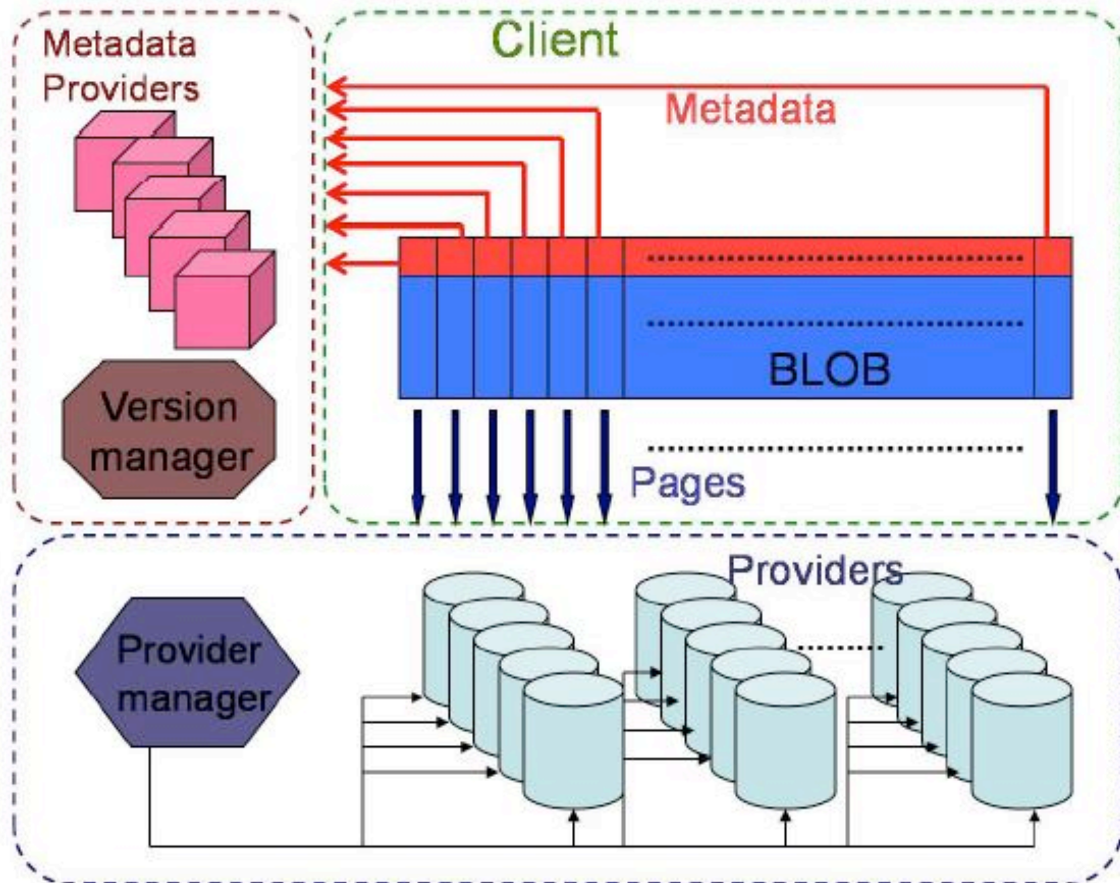
The data is stored in binary large objects, called *blobs*, which are divided in blocks of fixed size, called *pages*. A page can be smaller than 1 MB. When users requests a write or append operation, a new page is added to the system and a new metadata entry is created.

Pages generated by write/append requests are stored on *data providers*. The number of data providers is not fixed, it is changed dynamically, depending on the operations done by clients.

There is a *data provider manager* where the data providers register. It has to maintain information about the storage space and to decide on which provider the new page will be written.

All the file versions are stored on the system, so this means that BlobSeer has components that store information about the location of each page and manage update requests. These components are: *metadata provider* and *version manager*. The last one is the most important component, because it assigns file versions and guarantees total ordering and atomicity of the updated data.

The interaction between these modules can be seen bellow:



Interactions between the version manager, clients and provider manager

As it is said in [2], all blob operations are initiated by clients, which can access the same blob in a concurrent way. The support for concurrent operations is enhanced by storing pages belonging to the same blob on multiple storage providers. Metadata providers host the metadata associated to each blob. The version manager deals with the serialization of concurrent operations like write or append. The provider manager keeps track of all storage providers from the system.

3.2. What a client can do

As I said above, now, a client can do three operations: read, write and append.

Read requests

In the first step, the client will contact the version manager to get the version of the corresponding blob. If the version is available, the client will contact the metadata provider, from where he retrieves the metadata associated with the pages of the segment corresponding to the requested version. In this way, the client will gather all the metadata and after this step, he will contact all the providers that store the requested pages.

Write and append requests

When a client has to write information, first he has to request the provider manager the list of all providers for each page that exists in the blob segment that have to be written. After he receives this information, he has to request in parallel rights to write the pages in the wanted segment. Each provider processes the request and if he agrees, sends an acknowledge message to client. After he receives acknowledges from all the providers, he asks the version manager a new version id for the new data that will be written. After this step, metadata for the new information is generated using the version number given by the version manager. When metadata is generated successfully, the client sends a success message to version manager.

The append operation is similar with writing; the only difference is that the client will specify the offset where the new data is written.

In the future, also the client will be able to register, login the system, deploy services, see the file versions and delegate rights. This features will be developed by other students at their diploma project and integrated in the final system.

3.3. Security and malicious clients

The main security problem is represented by the fact that BlobSeer has no authentication mechanism or any way to distinguish the users. Each client accesses the information in the same way. Another hole in the security system is that there is no way of certifying the users.

Because the system is created to store and write big amount of data, the developers implemented an efficient way to enable concurrency in parallel writes. In this case, we may have problems with system integrity and consistence of the actions done by clients.

The malicious clients in our system are defined as clients that want to modify the system's stability and consistency. They are classified from the way they create writing inconsistency. So, we can speak about: *WriteNoPublish clients*, *PublishNoWrite* clients and *IncorrectWrites* clients.

WriteNoPublish clients

They write data to one or more providers, but they do not contact the version manager to publish the version of the written data. In this case, the system will be fulfilled with useless information and the bad point is that nobody from the system will know about it existence, because the version number is not published. In these cases, if a big number of clients are involved, the storage space will be filled up and the providers will not have enough space to host useful data.

PublishNoWrite clients

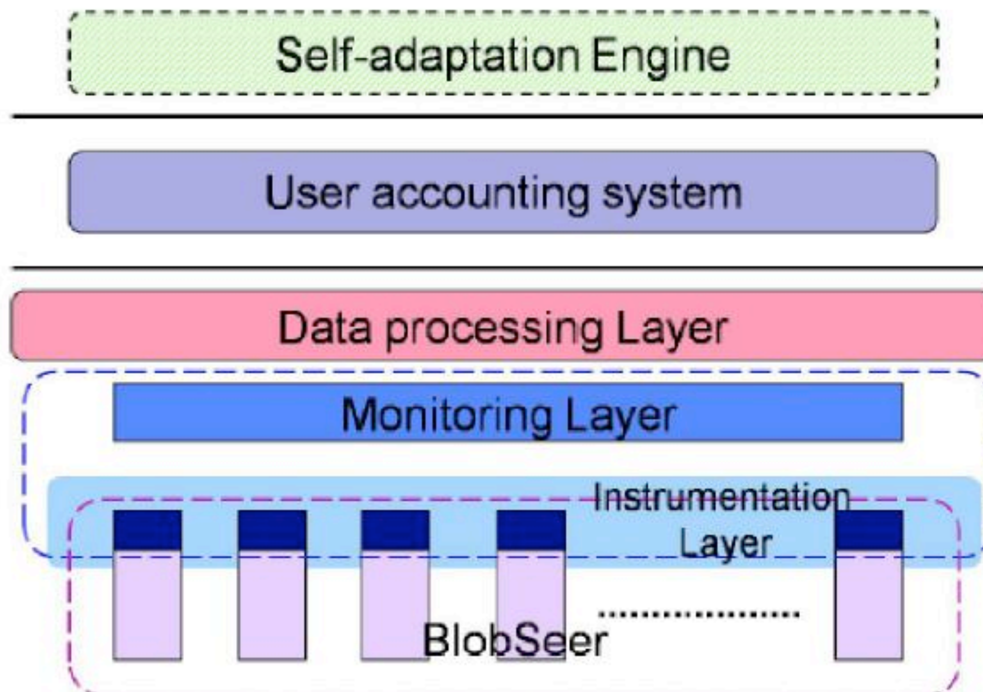
They publish the version and create the metadata tree, but they write nothing actually. This kind of attack will affect the consistency of read information. The system will behave in this way: when the version is requested, a metadata tree is created. The

affected blob will point to the previous version. When a client wants to read information from that blob, BlobSeer will try to take the parts of the blob that were not covered by the last write from the previous version, the one that doesn’t actually have any blob pages associated to it. In this case, we will have read inconsistency. These kinds of attacks affect the current written version and the following ones.

<u>IncorrectWrites clients</u>

The clients from this category write different sizes of the pages to providers and version managers. In this case, we have also read and write inconsistency. The versioning systems and the storage are affected and the main consequence is that the information written in this context will not be properly accessed.

A monitoring layer to enable introspection and automatic behavior is considered to be the first step in order to solve these security problems. This layer will be created above the instrumentation layer. The architecture can be seen in the figure bellow:



User accounting module in BlobSeer

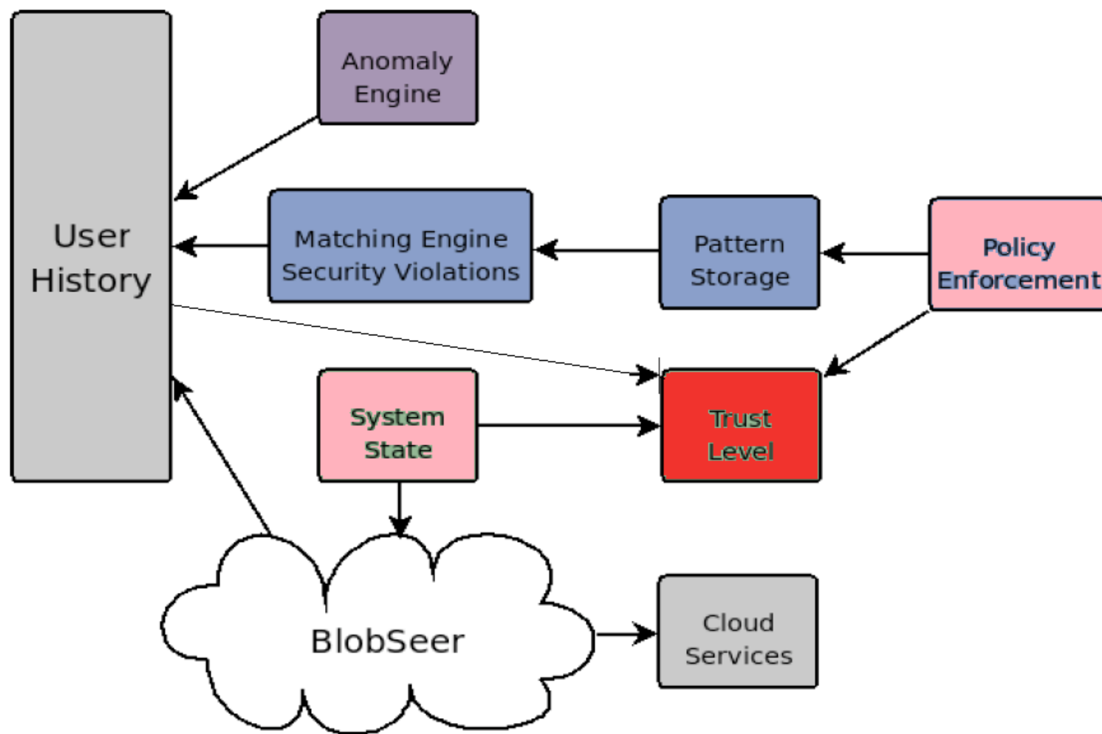
The monitoring layer will be useful to develop a malicious client detection module, where we give a mark to each client according to his history and activity. It will use MonALISA monitoring framework.

Also, to solve the problem of differentiating clients we will implement some access patterns and enforce adaptive security rules. We will distinguish the administrator from the ordinary clients. The administrator will define violation patterns and will receive system alerts when a user breaks the security policy already defined. When a system alert is generated, the administrator may run some scripts to ban access or remove writing rights to the malicious client or he may configure the system to do this automatically.

4. Malicious clients detection module

4.1. Theoretical issues

Malicious clients' detection module is in fact a layer in BlobSeer system. Its main functions are to solve the security hole and to administrate the user logins and requests. It is very complex and interacts with other system components that are described bellow:



Malicious clients' detection architecture

User history and *Cloud services* (the components drawn with gray) are not part of the framework, but the module will interact with them. More precisely, the information about user (his activity and his past behavior) is taken from *User history* module and

Cloud services functionalities are based on the actions taken by the *malicious clients' detection* layer.

My tasks where to develop the *Trust level module* (drawn with red) and to link it with the two ones: *Policy Enforcement* and *System State* (drawn with pink).

System State shows the system state, which is influenced by the density of requests, the load of providers on the version manager and the global load of the system.

Policy Enforcement contains predefined policies. They are present in any BlobSeer distribution. They are based on control the access rights of existing clients, details about the quality of service for the users. These basic policies can be divided into three categories: limit the read/write request to a time interval, limit the number of new versions into a time interval to a certain number and limit the access to data, to certain blobs from the system. Besides these, the administrator can define any custom policies, using an XML file format that is integrated on the fly by the detection module.

Trust Level receives the history of a certain user, the system state and the policy broken (or not, depends on case) and computes a mark for the given user. This module focuses on logs analysis, from where we can see three main cases: analysis of broken policies, protocol breach and abnormal client activity.

The trust level is computed using a complex formula based on the analysis made in the three cases, the system state and the user past:

$$\sum A_i * Age(time_i) * SystemState(i)$$

In this formula:

- Ai means the coefficient of the action taken by the user in the system; it has integer values between 0 and 50. A big coefficient indicates a very bad action and a small one an innocent action. We receive the actions from Monalisa monitoring module. And we extract the value from the database, from policy_events table, where each action has an importance (high –j, low –l and medium –m) and a value (action_value) [see annex 5].

- *Age ($time_i$)* is a time function, used to balance the activity of the client.

$$Age(time_i) = \frac{1}{currentTime - time_i}$$

In this formula, *currentTime* means the time when we compute the trust and $time_i$ the time when the action was taken. Both *currentTime* and $time_i$ are measured in milliseconds.

This term shows us that an action done a long time ago has a less importance than a recent action.

- *SystemState (i)* is a function that returns the system state at the moment we compute the trust. First we, determine a number that represents the state at that moment:

$$systemStateValue = \frac{(load5 + (100 - freeMemory))}{200}$$

Where *load5* means the load of the processor and *freeMemory* the free memory at the moment we compute the *systemStateValue*. We get these data from the database, system_state_134.125.124.123 table. This table contains data taken from the system by MONalisa agent. After we compute this, we can define the *SystemState(i)* function formula, as you can see bellow:

$$SystemState(i) = \begin{cases} 1, systemStateValue \leq 0.2 \\ 1 + systemStateValue, otherwise \end{cases}$$

So, we can see that the *SystemState (i)* has values between 1 and 2. If we have a stable system, we consider the *systemStateValue* less or equal to 0.2. In this case, the *SystemState* function is 1. So, we consider that the state does not influence the result of the trust. If we have a system with problems, the systemStateValue is bigger than 0.2, and the function is bigger than 1, so the trust level of that client will be influenced by the system.

The mark given to a client has values between 0 and 100. Using this score, we define four types of clients: *malicious, bad, with normal activity* and *good*. Those considered malicious have a trust between 0 and 10, those who are bad 10 – 20, those considered with normal activity (we can trust them, because they did not break any security policy) have a trust level between 20 and 30 and the good ones have a trust bigger than 30.

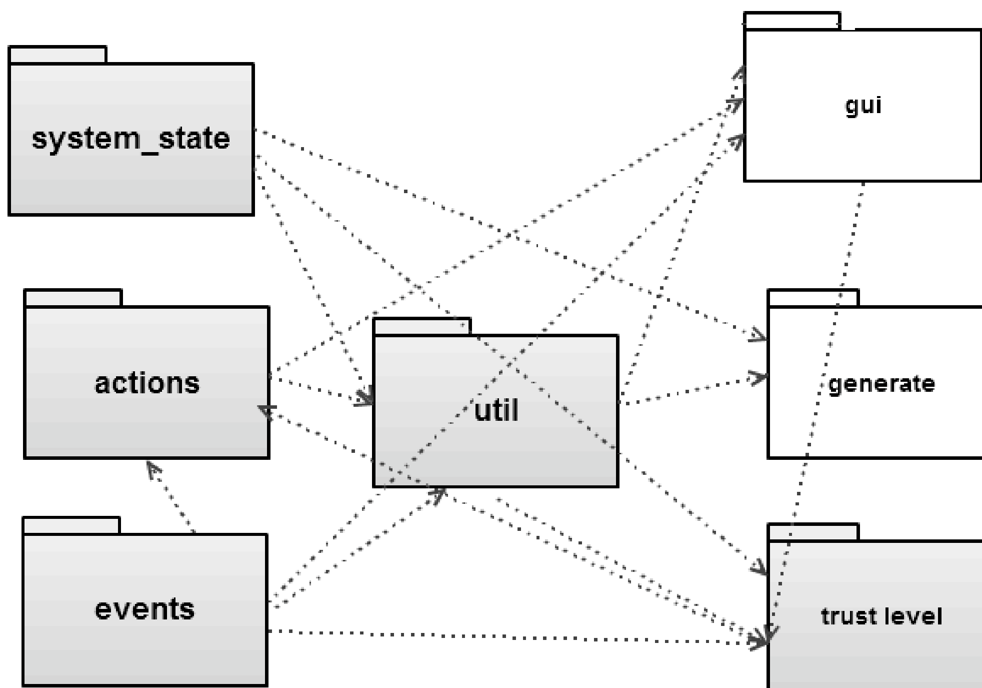
We can see that the trust level has small values, generally between 0 and 10, because it is dynamically changed, depending on user activity and the system state. In time, if the user has a right activity and he does not damage the system, his trust will increase, and he will go in a better category and he will obtain more rights. If the user does wrong actions, his trust will decrease, sometimes dramatically, and he will decay a worse user category.

So, the trust computed will be used by the system to punish or recompense the client. All these changes will be written in log files, which data will be taken by the PolicyEnforcement module.

4.2. Implementation

The application is, in fact, a simulation of a malicious clients' detection module. It is composed of seven packages: *actions, events, generate, gui, system_state, trust_level, util, util.db.*

You can see bellow the package diagram:

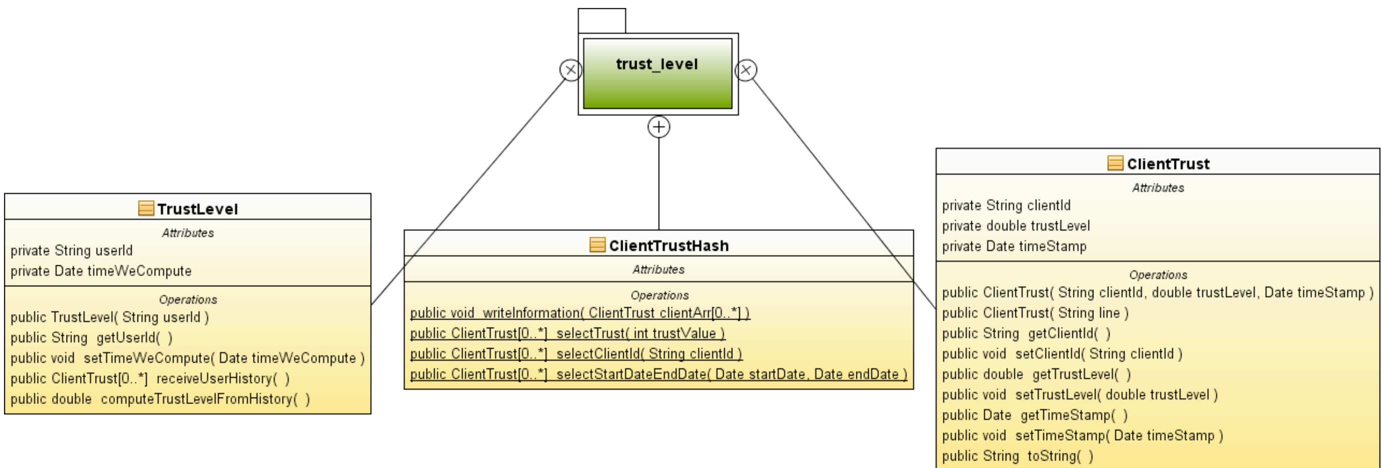


Package diagram

The most important packages for our application are described bellow.

1) trust_level package description

For our application, the most important is trust_level package. It is composed of 3 classes: ClientTrust.java, ClientTrustHash.java and TrustLevel.java.



In TrustLevel.java, we compute the trust level for a given user at a certain time.

It has two private members: userId and timeWeCompute, that are set in the class constructor and four methods. From these, important for our application are:

- receiveUserHistory()
  - Get information about the client from trust_clients.txt log file
  - This information consists of an array where are stored ClientTrust objects, that contains the client id, the trust level and the date when the trust was computed
  - This function is useful when the administrator wants to see the client ‘s trust history in the system.
- computeTrustLevelFromHistory()
  - Here we compute the trust for the client
  - We use the formula described at the beginning of this chapter
  - After the trust is computed, we write the trust in the trust_clients.txt

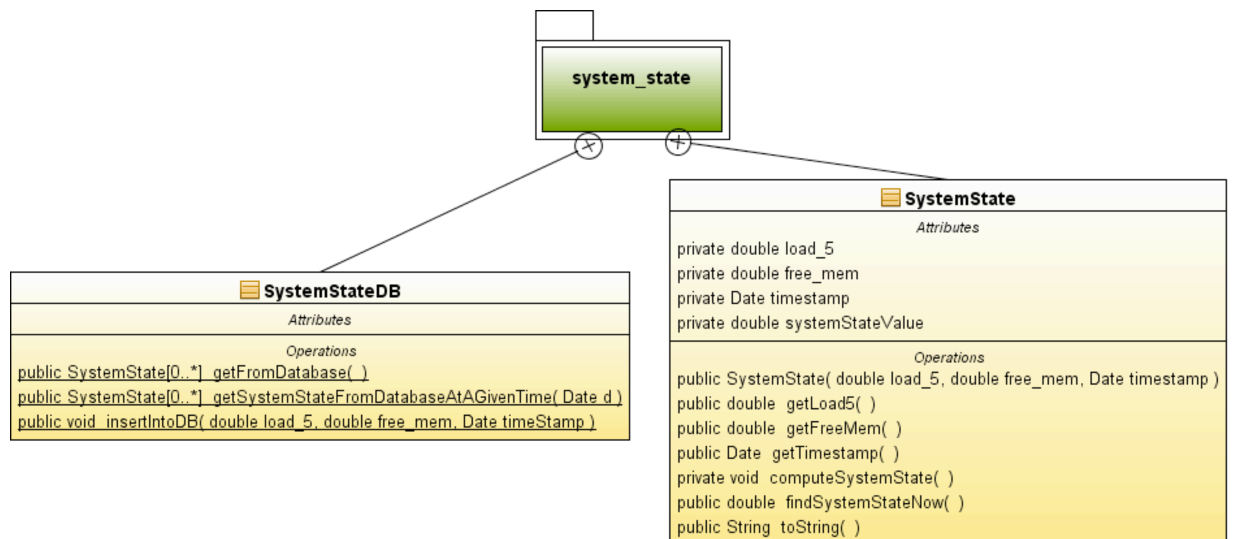
In ClientTrust.java, we store the client id, the trust computed and the data when we compute the trust.

In class ClientTrustHash.java, we have static methods used for:

- Write information in the trust log file
- Select information from trust log file about a client
- Select from trust log file all the clients that have a given trust level
- Select from trust log file all the clients that have the trust computed between two given dates

2) system_state package description

The package is used to take information about the system state from the database and to compute the *systemState* function value, using the data taken from the database. It is composed of two classes: SystemState.java and SystemStateDB.java. Bellow, you can see the interaction the two classes.



In SystemStateDB class, we select information about the state from the database (which is taken from the monitoring module interaction), using the methods:

- getFromDatabase
  - selects all the information from the database

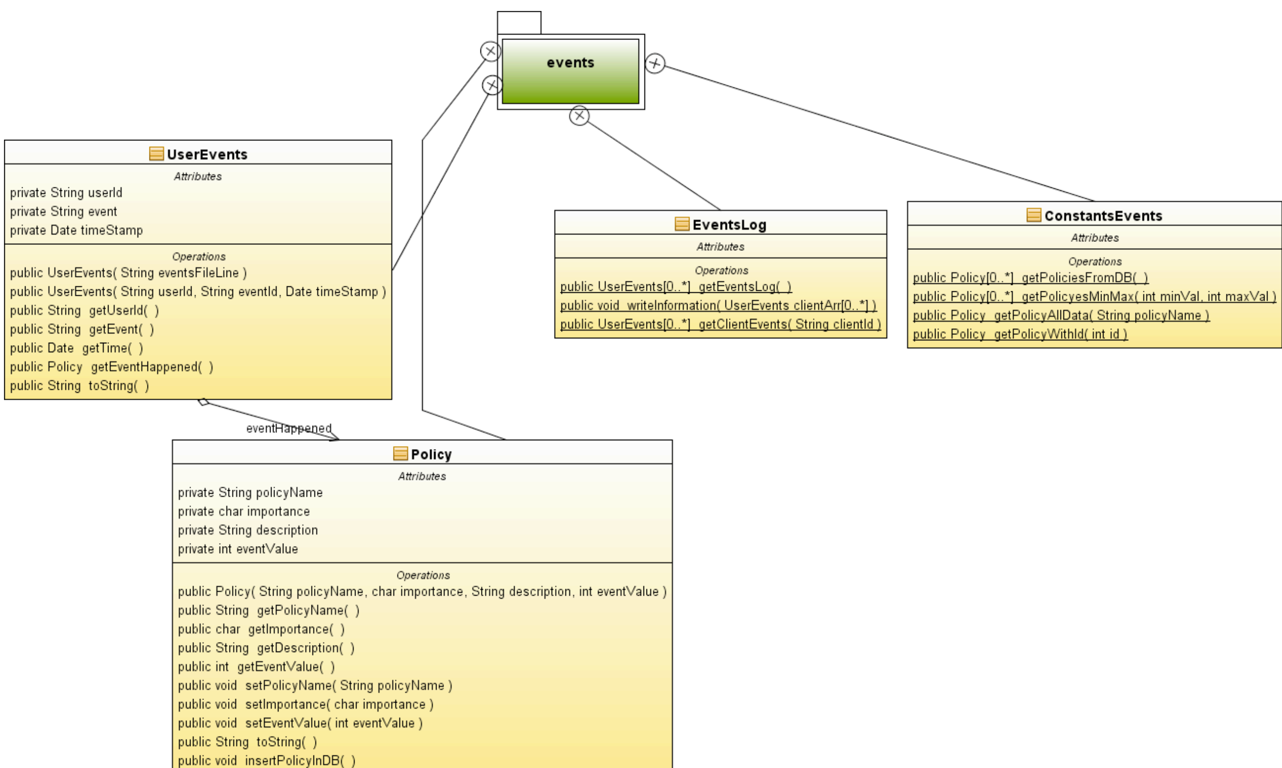
- getSystemStateFromDatabaseAtAGivenTime
    - selects all the system state information at a given time

Also, in this class, we write information in the database. The monitor module will use this function to populate the database.

In SystemState class, we store information about the system state: the load, the free memory, the time we received these information from the system. Also, here, we compute the systemStateValue, a number that is used to determine the systemState function value.

3) <u>events package description</u>

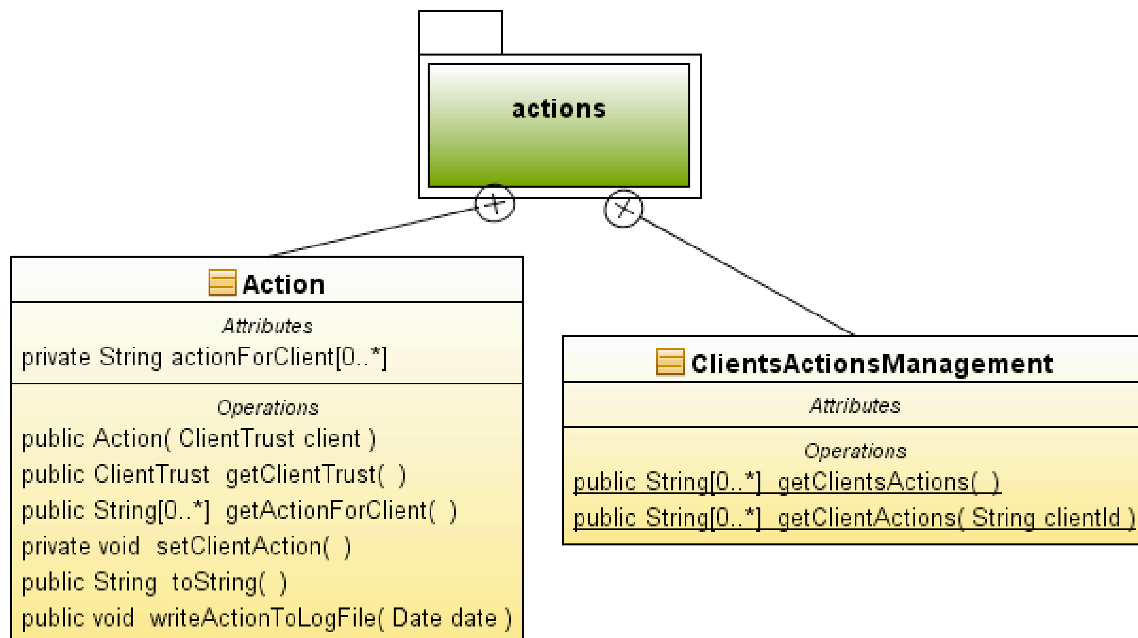
In this package, we process data about clients activity (events happened in the system). It is composed of four classes: ConstantsEvents.java, EventsLog.java, Policy.java and UserEvents.java. Bellow, you can see the package diagram:



UserEvents class stores information about users, the actions done by they and the time when those actions happened. In Policy class, are stored the event name, the importance, the description of the event and the value. The events may be of several types, including the policy violation. When an event is of this type, it will receive a negative the value. In the database are stored the positive values for each action that can be done in the system. In ConstantsEvents class, are defined a couple of static methods, that select information about the events (description, importance and value) from the database. In fact, it is a wrapper over the policy_events table. In class EventsLog, we have methods for writing and reading information in/from the clients_activity.log log file.

4) actions package description

This package processes the actions that will be taken for a user at a certain moment. It consists of two classes: Action and ClientActionsManagement. Bellow, you can see the package diagram:



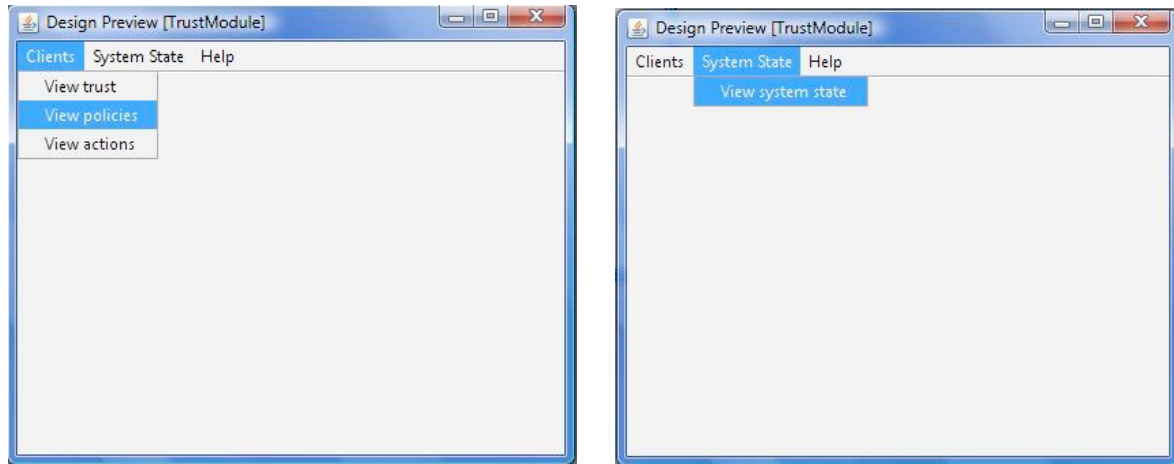
In the first class, Action, we set the system attitude after an user activity. The most important methods are:

- setClientAction()
  - using the client’s trust, we set a couple or rights
  - there are defined five possible rights for a client:
    - the default rights are: append, read, write, login and register
  - if the client becomes a malicious one, he will lose some of these rights and he will be able only to login and read data
  - if the client has a good activity, he will receive more rights, like: file versioning
  - if the client has a very good activity after a long time, he will be able to deploy services
  - if the client doesn’t have any bad activity, he will be also able to delegate some of his rights
- writeActionToLogFile(Date date)
  - writes the rights set to the client in the clients_actions.log log file.

In ClientsActionsManagement.java, there are defined a couple of methods to read and write data in the clients_actions.log log file.

In util and util.db, there are defined methods for database interactions, constants, methods for processing time values.

In gui package, there are classes that generate the user interface for the admin (this feature was created in addition to the requirements of the project), where he will see the history of each client from the system and the system state. For each client, he will see the trust level at different moments of time, the policies adopted for those clients. Bellow, you can see how the user interface looks:



The graphical interface will have a menu composed of three tabs: Clients, System State and Help. From “Clients” menu, the administrator can select information about one or more clients trust, policies and actions. From “System State” menu, the administrator can see the see information about the system state at certain moments. From the “Help” menu, they will see information about the application usage.

4.3. Interaction with other BlobSeer modules. Input and output data format

This module will be integrated in the BlobSeer system. The module will need the clients, their activity, the time when they did that activity and the system state. In this situation, it will take the input data from:

- database:
  - from the system_state_134.125.124.123 table, where are put the processor load, memory and timestamp; this table is populated from the monitor module
  - from the policy_events table, a static table, that contains all possible events from the system with their descriptions and values
- log files:
  - clients_activity.log, where are stored the client id, the activity done by him and the time when he did that activity

The output will be the trust and the rights that a client may have. These information are stored in two ouput files:

- clients_actions.log, where are stored the date, client id, trust level and the possible actions that this client is allowed to do
- trust_clients.txt, where are stored the client id, trust and the date when that trust is computed

The information stored in these files is used by other BlobSeer modules.

More information about these files you may see in annex 6.

5. Experimental results and conclusions

I tested this module on my computer. I used generated data to simulate clients' activity in this system. Bellow you may see an example of test.

For this input data:

- system state:

load_5 double precision	free_memory double precision	timestamp date
6.03343372202874	14.317013445935	2010-06-28
61.25385216101	92.0174462991146	2010-06-26
62.2691147039488	48.8070060793344	2010-06-02
8.4909224783929	5.79978978811791	2010-06-11
53.995351041035	10.365923747017	2010-06-11
16.3624732135802	54.0726153015094	2010-06-25
16.2479672991466	81.4946126034568	2010-06-17
94.5860240157347	79.2615191982687	2010-06-08
91.6106309896565	71.049440936066	2010-06-13
26.5100442866724	83.4025125094951	2010-06-18
6.16450998864665	83.8093897706411	2010-06-21
91.1715397797812	2.70879253992426	2010-06-15
92.5326931586471	54.4649812525954	2010-06-23
30.5388897940437	53.8330432073095	2010-06-03
20.2204529195478	99.9583109008345	2010-06-11
77.5553743104126	23.1695264411109	2010-06-29
30.8853154876415	89.3301334190607	2010-06-19
79.180754765901	3.41623464958717	2010-06-23
86.2152793790969	31.4434537855717	2010-06-01
87.1454233903829	17.1509242432647	2010-06-18

- clients activities [clients_activiy.log file]:

```
client11,illegal user,2010-06-28
client5,write no publish,2010-06-26
```

client4,illegal user,2010-06-02
client12,append data,2010-06-11
client5,read pages without requesting them from metadata providers,2010-06-11
client8,delegate rights,2010-06-25
client12,delegate rights,2010-06-17
client13,crowling,2010-06-08
client7,write limit exceded,2010-06-13
client1,delegate rights,2010-06-18
client6,crowling,2010-06-21
client2,client without activity for a long period of time,2010-06-15
client6,large number of versions for the same data,2010-06-23
client8,access older file versioning,2010-06-03
client7,read data,2010-06-11
client9,register in the system,2010-06-29
client10,write no publish,2010-06-19
client12,write data to a blob,2010-06-23
client6,write limit exceded,2010-06-01
client3,illegal user,2010-06-18

We receive this output:

- clients allowed actions in the system [clients_actions.log file]:

2010-07-06 : client11 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client5 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client4 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client12 , 2.4825824679710863E-8 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client8 , 1.93365473960193E-8 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client13 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client7 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client1 , 9.335339018615438E-9 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client6 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client2 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client9 , 1.1579375833299138E-8 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client10 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client3 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;

- clients trust [trust_clients.txt file]:

client11,0.0,2010-07-06

client5,0.0,2010-07-06
client4,0.0,2010-07-06
client12,2.4825824679710863E-8,2010-07-06
client8,1.93365473960193E-8,2010-07-06
client13,0.0,2010-07-06
client7,0.0,2010-07-06
client1,9.335339018615438E-9,2010-07-06
client6,0.0,2010-07-06
client2,0.0,2010-07-06
client9,1.1579375833299138E-8,2010-07-06
client10,0.0,2010-07-06
client3,0.0,2010-07-06

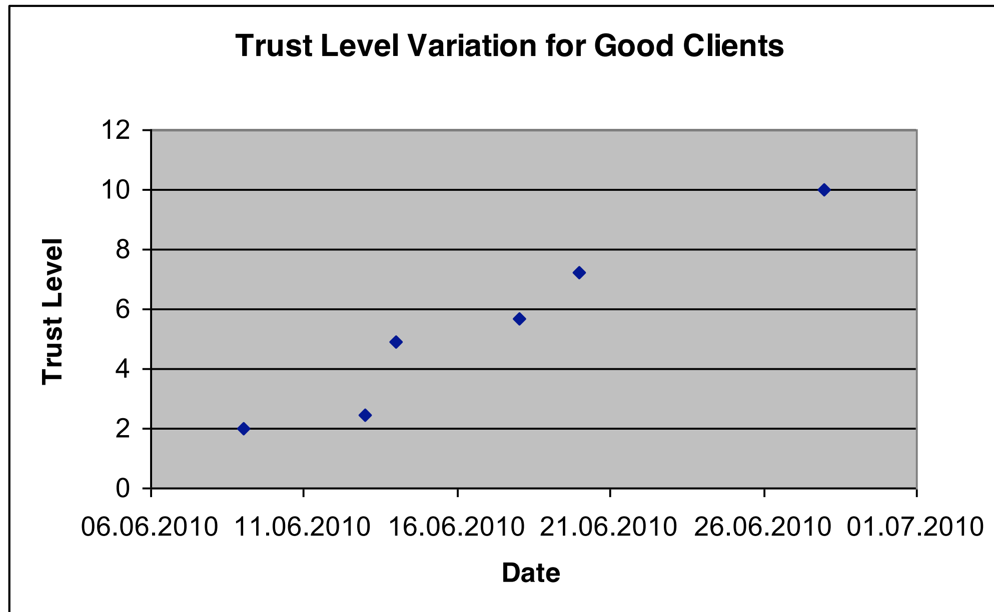
So, we compute the trust level of twenty clients on 06-07-2010. These clients were new in the system. If you may see above, these clients did different activities. Ones of those, like client11, client5, client4, client13, client7, client6, client2, client10 and client3 did harmful things, and when we compute the trust for them, they receive a negative trust, which was rounded to zero. Because they are at their first deviation, they will have only their default rights, as you may see in clients_actions.log output file. For e.g., user client10 may register, login in the system, read, write and append data. For other users - client9, client1, client12 and client8, we can see that they have positive trust, because they done normal actions, that didn't harm the system stability. Because they are at their first activity, they have a little trust value. So, after a short period of time when they activate in the system, they are in the first category with the first ones.

*

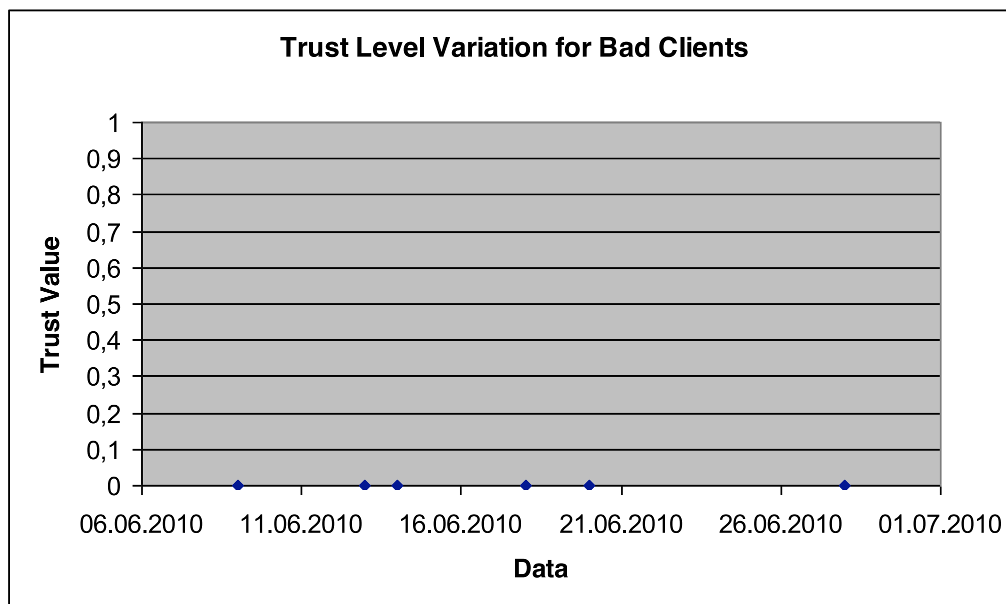
* *

After collecting the results, I saw that the client trust level varies in an undefined function form. If he has only malicious activities, his trust will became in short time zero. If he has only activities that are not included in the policy violation area, his trust will increase. Because you see that the trust is a cumulative function, in the last case, we cannot say that his trust will increase linearly. The middle case is when the client has also policy violation activities, but also activities that are not bad for the system. In this case, the trust will fluctuate, like a wave function, with unequal waves.

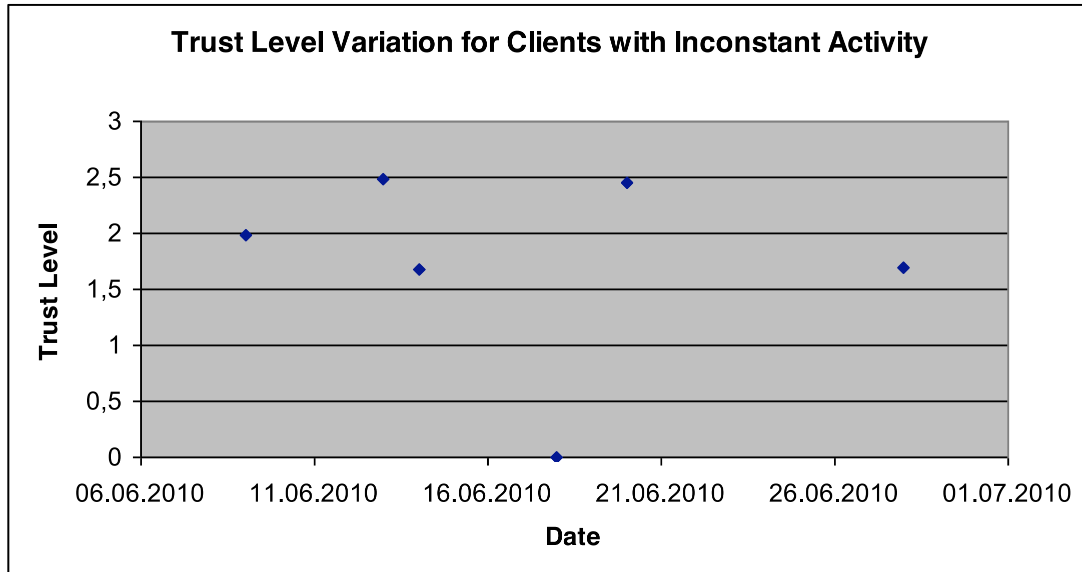
Bellow, you may see some graphics that sustain my conclusions.



Trust Level variation for good clients



Trust Level variation for bad clients



Trust Level for a client with good and also suspicious activity

6. Future work

In the future, we will integrate the module in BlobSeer system and we will link with the other modules described in this thesis. We will perform much more tests with a bigger number of clients and we will try to get a better formula (if possible) for the trust. Also, we can improve this module, giving the administrator the opportunity to choose from a set of rights that he may give to the system clients. Also, after the services module will be done, we will add more possible actions for our clients and also more policies and rights for them.

References

[1]Alexandra Carpen-Amarie, Jing Cai, Luc Bougé, Gabriel Antoniu, Alexandru Costan, *Monitoring the BlobSeer distributed data-management platform using the MonALISA framework*, Research Report RR-7018, INRIA, 2009.

[2]Alexandra Carpen-Amarie, Jing Cai, Luc Bougé, Gabriel Antoniu, Alexandru Costan, *Bringing Introspection Into the BlobSeer Data-Management System Using the MonALISA Distributed Monitoring Framework*, "International Workshop on Autonomic Distributed Systems, 2010.

[3]Lik Mui, *Computational Models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks*, Massachusetts Institute of Technology, 2002.

[4]B. Nicolae, G. Antoniu, L. Bougé, BlobSeer: *How to enable efficient versioning for large object storage under heavy access concurrency. In Data Management in Peer-to-Peer Systems*, St-Petersburg, Russia, 2009.

[5]Cristina Băsescu, Catalin Leordeanu, Alexandru Costan, Valentin Cristea, *Towards Malicious Client Detection in Blobseer*, Bucharest, 2010.

[6] http://blobseer.gforge.inria.fr [10 April 2010]

[7] Eric Knorr, Galen Gruman, *What cloud computing really means,* in http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031 [7 May 2010]

[8] http://monalisa.caltech.edu/monalisa.htm [8 May 2010]

[9] B. Nicolae, "High Throughput Data-Compression for Cloud Storage," in *Proc. 3rd International Conference on Data Management in Grid and P2P Systems (Globe'10)*, Bilbao, Spain, 2010

[10] B. Nicolae, "BlobSeer: Efficient Data Management for Data-Intensive Applications Distributed at Large-Scale," in *Proc. 24th IEEE International Symposium on Parallel & Distributed Processing: Workshops and Phd Forum (IPDPS '10)*, Atlanta, USA, 2010, pp. 1-4.

[11] B. Nicolae, D. Moise, G. Antoniu, L. Bougé, and M. Dorier, "BlobSeer: Bringing High Throughput under Heavy Concurrency to Hadoop Map/Reduce Applications," in *Proc. 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS '10)*, Atlanta, USA, 2010, pp. 1-11.

[12] B. Nicolae, G. Antoniu, and L. Bougé, “Enabling High Data Throughput in Desktop Grids Through Decentralized Data and Metadata Management: The BlobSeer Approach," in *Proc. Euro-Par '09*, Delft, The Netherlands, 2009, pp. 404-416.

[13] Krissi Danielson, Distinguishing Cloud Computing from Utility Computing, 2008, in http://www.ebizq.net/blogs/saasweek/2008/03/distinguishing_cloud_computing/

[14] Buyya, Rajkumar; Chee Shin Yeo, Srikumar Venugopal, *Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities*, Department of Computer Science and Software Engineering, University of Melbourne, Australia. pp. 9, 2008 in http://www.gridBus.org/~raj/papers/hpcc2008_keynote_cloudcomputing.pdf

[15] Resnick, P., Zeckhauser, R., Friedman, E., Kuwabara, K., *Reputation Systems*, 2000, in http://presnick.people.si.umich.edu/papers/cacm00/reputations.pdf

[16] D. Quercia, S. Hailes, L. Capra, *Lightweight Distributed Trust Propagation* in http://www.cs.ucl.ac.uk/staff/d.quercia/publications/quercia07lightweight.pdf

[17] R. Guha, R. Kumar, P. Raghavan, A. Tomkins, *Propagation of Trust and Distrust* in http://www.ra.ethz.ch/CDstore/www2004/docs/1p403.pdf

[18] Lazzari, Marco, *An experiment on the weakness of reputation algorithms used in professional social networks: the case of Naymz*, Proceedings of the IADIS International Conference e-Society 2010. Porto, Portugal, 2010 in http://dspace-unibg.cilea.it/handle/10446/572

[19] ***, *The Art of Monitoring with MonALISA*, Science Grid This Week, 2008, in http://www.interactions.org/sgtw/2005/0831/monalisa_more.html

[20] http://gridcafe.web.cern.ch/gridcafe/demos/monalisa.html

Annexes

Annex 1: BlobSeer Installation

BlobSeer is designed for Unix systems, so we choose to install it on a Ubuntu distribution. In this section, I will describe the installation steps.

Step 1:

First of all, before you start to install BlobSeer, you have to check if on your operating system you have the following applications: gcc, java (it is better to have jdk 1.6), g++, cmake and svn. You can check it, using the command bellow (for e.g. we search if cmake is already installed):

```
sudo apt-cache search cmake
```

Step 2:

After you installed them, you have to download BlobSeer from http://blobseer.gforge.inria.fr/doku.php?id=main:download (in fact you have to checkout the BlobSeer repository). The last release version you will find here http://blobseer.gforge.inria.fr/doku.php?id=news . At this step, you will have to run this command:

```
svn checkout svn://scm.gforge.inria.fr/svn/blobseer/tags/release-Version
```

For our application, we used version 0.3.3 and the command looks like this:

```
ana@ana-desktop:~$ svn checkout svn://scm.gforge.inria.fr/svn/blobseer/tags/rele
ase-0.3.3
A    release-0.3.3/test
A    release-0.3.3/test/appender.cpp
A    release-0.3.3/test/test.cpp
A    release-0.3.3/test/basic_test.cpp
A    release-0.3.3/test/multiple_readers.cpp
A    release-0.3.3/test/create_blob.cpp
A    release-0.3.3/test/worker_mapred.cpp
A    release-0.3.3/test/streamer.cpp
A    release-0.3.3/test/multiple_writers.cpp
A    release-0.3.3/test/CMakeLists.txt
```

Step 3: Modify .bashrc file from your profile

You will add to .bashrc the path to your java home and ld_library_path, as you can see bellow:

- for Java : find where you have already installed java:

```
ana@ana-desktop:~/Downloads/boost_1_42_0$ locate jdk
/usr/lib/jvm/java-6-sun-1.6.0.15/jre/lib/servicetag/jdk_header.png
/usr/lib/jvm/java-6-sun-1.6.0.15/lib/visualvm/profiler3/lib/deployed/jdk15
/usr/lib/jvm/java-6-sun-1.6.0.15/lib/visualvm/profiler3/lib/deployed/jdk16
/usr/lib/jvm/java-6-sun-1.6.0.15/lib/visualvm/profiler3/lib/deployed/jdk15/linux
/usr/lib/jvm/java-6-sun-1.6.0.15/lib/visualvm/profiler3/lib/deployed/jdk15/linux
/libprofilerinterface.so
/usr/lib/jvm/java-6-sun-1.6.0.15/lib/visualvm/profiler3/lib/deployed/jdk16/linux
/usr/lib/jvm/java-6-sun-1.6.0.15/lib/visualvm/profiler3/lib/deployed/jdk16/linux
/libprofilerinterface.so
```

In our case, the path is /usr/lib/jvm/java-6-sun-1.6.0.15/bin

You will append .bashrc file as bellow:

```
ana@ana-desktop:~$ gedit .bashrc
ana@ana-desktop:~$ source .bashrc
ana@ana-desktop:~$ echo $JAVA_HOME
/usr/lib/jvm/java-6-sun-1.6.0.15/bin
ana@ana-desktop:~$ tail .bashrc
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi

#instructions added to bashrc in order to install BlobSeer
#export java home
export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.15/bin
```

Set LD_LIBRARY_PATH, where all libraries for your project will be:

export LD_LIBRARY_PATH=$HOME/deploy/lib

You will do the same steps at a). The results will be as bellow:

```
ana@ana-desktop:~$ pwd
/home/ana
ana@ana-desktop:~$ gedit .bashrc&
[1] 2414
ana@ana-desktop:~$ source .bashrc
[1]+  Done                    gedit .bashrc
ana@ana-desktop:~$ tail .bashrc
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi

#instructions added to bashrc in order to install BlobSeer
#export java home
export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.15/bin
#export library path
export LD_LIBRARY_PATH=$HOME/deploy/lib
ana@ana-desktop:~$ echo $LD_LIBRARY_PATH
/home/ana/deploy/lib
ana@ana-desktop:~$
```

Tricks & tips:

- In .bashrc there are not allowed spaces
- To update the settings, you should do source .bashrc in your console and the changes will be taken
- To check if your export path is ok, use echo $PATH_EXPORTED (in our case path_exported may be JAVA_HOME)

Step 4:

BlobSeer needs external dependencies: Boost, Libconfig and Berkley DB.

Step 4.1: Boost installation

First, you have to download version 1.42 for Unix from http://www.boost.org/ . If you choose an older version, the BlobSeer installation will fail. You should follow the instructions bellow:

./bootstrap.sh --prefix=$HOME/deploy --with-libraries=system,thread,serialization,filesystem,date_time -libdir=$HOME/deploy/lib

```
ana@ana-desktop:~/Downloads/boost_1_42_0$ pwd
/home/ana/Downloads/boost_1_42_0
ana@ana-desktop:~/Downloads/boost_1_42_0$ ./bootstrap.sh --prefix=$HOME/deploy --with-libraries=system,thread,serialization,f
ilesystem,date_time -libdir=$HOME/deploy/lib
Building Boost.Jam with toolset gcc...
```

After that, follow the instructions written on the screen, and run bjam:

./bjam

You should install the libraries and run in the console:

./bjam install

Trips & tricks:

- To check if all the libraries are correctly install, run:

  ls $HOME/deploy/lib

- *All necessary libraries will be installed here.*

Stept 4.2: Libconfig installation

Libconfig is a library for C/C++ files. It can be downloaded from http://www.hyperrealm.com/libconfig/. After this, you extract the content and you should run:

./configure --prefix=$HOME/deploy && make && make install

After this, you should check if the libraries are correctly installed (see *Trips & tricks* section).

Step 4.3: Berkley DB installation

You will download the package from http://www.oracle.com/technology/software/products/berkeley-db/index.html and you will choose the version for Linux.

In build_unix, you will run:

```
../dist/configure  --prefix=$HOME/deploy --enable-cxx
make
make install
```

After you complete these steps, check if all libraries are installed. You should have an output like the bellow one:

```
ana@ana-desktop:~$ ls /home/ana/deploy/lib
libboost_date_time.a                 libconfig++.la
libboost_date_time.so                libconfig.so
libboost_date_time.so.1.42.0         libconfig++.so
libboost_filesystem.a                libconfig.so.8
libboost_filesystem.so               libconfig++.so.8
libboost_filesystem.so.1.42.0        libconfig.so.8.1.2
libboost_serialization.a             libconfig++.so.8.1.2
libboost_serialization.so            libdb-5.0.a
libboost_serialization.so.1.42.0     libdb-5.0.la
libboost_system.a                    libdb-5.0.so
libboost_system.so                   libdb-5.so
libboost_system.so.1.42.0            libdb.a
libboost_thread.a                    libdb_cxx-5.0.a
libboost_thread.so                   libdb_cxx-5.0.la
libboost_thread.so.1.42.0            libdb_cxx-5.0.so
libboost_wserialization.a            libdb_cxx-5.so
libboost_wserialization.so           libdb_cxx.a
libboost_wserialization.so.1.42.0    libdb_cxx.so
libconfig.a                          libdb.so
libconfig++.a                        pkgconfig
libconfig.la
```

Step 5: BlobSeer installation

Now you are ready to install BlobSeer, because all the dependencies are already on your operating system.

Your home should look like in the printscreen bellow:

```
ana@ana-desktop:~$ ls
deploy    Documents  examples.desktop  Pictures  release-0.3.3  Videos
Desktop   Downloads  Music             Public    Templates
```

Firstly, you have to build the makefiles, using the command:

cmake -G "Unix Makefiles"

```
ana@ana-desktop:~/release-0.3.3$ cmake -G "Unix Makefiles"
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Found Boost: /home/ana/deploy/lib/libboost_system.so;/home/ana/deploy/lib/li
bboost_thread.so;/home/ana/deploy/lib/libboost_serialization.so;/home/ana/deplo
y/lib/libboost_filesystem.so;/home/ana/deploy/lib/libboost_date_time.so
-- Found config++: /home/ana/deploy/lib/libconfig++.so
-- Found Berkeley DB: /home/ana/deploy/lib/libdb_cxx.so
-- JNI headers not found. Java support will not be built.
-- Ruby headers not found. Ruby support will not be built.
-- Python headers not found. Python support will not be built.
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ana/release-0.3.3
```

After this, you can start to build BlobSeer, so you should run:

make

```
ana@ana-desktop:~/release-0.3.3$ make
Scanning dependencies of target provider
```

Step 6: Create public and private keys

Now, you have to create a pair of keys, in order to login in the system. For this, you have to follow the instructions bellow:

Step 6.1: Install ssh

```
ana@ana-desktop:~/release-0.3.3$ sudo apt-get install openssh-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Step 6.2: Create the public and private key pair

You should locate in your home directory, in .ssh directory. Here you should run the command, in order to generate the public and the private key:

ssh-keygen –t rsa

The output should be like the bellow one:

```
ana@ana-desktop:~/release-0.3.3$ cd $HOME/.ssh
ana@ana-desktop:~/.ssh$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ana/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ana/.ssh/id_rsa.
Your public key has been saved in /home/ana/.ssh/id_rsa.pub.
The key fingerprint is:
76:bf:0d:e0:85:95:07:3c:3f:81:19:62:43:17:2f:fd ana@ana-desktop
The key's randomart image is:
+--[ RSA 2048]----+
|          .=.==  |
|          . +==. |
|             +o+.|
|            o oo.|
|         S + .  .E|
|       . o +     |
|          . o    |
|             +   |
|            . .  |
+-----------------+
```

Step 6.3: Add the public key to authorized_keys file

authorized_keys file will be created in $HOME/.ssh directory. You can see how to this bellow:

```
ana@ana-desktop:~/.ssh$ pwd
/home/ana/.ssh
ana@ana-desktop:~/.ssh$ ls
id_rsa  id_rsa.pub
ana@ana-desktop:~/.ssh$ mv id_rsa.pub authorized_keys
ana@ana-desktop:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEApLT5MxqZabDQ+v3fn0SMAlmqOPy9YAmmUj3DkkNwSld
v41oP/7grqBaAireakWVzSsUMFt49LbSjm+/mmaTiZhYaKjlpgsDalimGx8XlpfmRsO31p4EZR8r/Ai
eZociPsVgBE9rJQWiCnam10dCxjXH6mckDCF612mjOc9hnlEwWydl6OTq8E2438q1PDXjrDZFjFfiSB
2WrXgjwJ0++9J8GEoI5ZVkjUZBsmTx48dAin6IInZfGq7PaTblwl0FYc7K7+l4v4c+0mL3aV1F2WlcJ
W15OLbE7E/ykpxQU8n4eki9Os3/s+HvTz5LbIvLnvz01lAHQTXLpRKpzR2zLpQ== ana@ana-deskto
p
```

Step 6.4: Create config file

config file will be created in $HOME/.ssh directory. The config file should contain: StrictHostKeyChecking no , which means that it is not necessary to login when you use BlobSeer.

```
ana@ana-desktop:~/.ssh$ pwd
/home/ana/.ssh
ana@ana-desktop:~/.ssh$ cat config
StrictHostKeyChecking no
ana@ana-desktop:~/.ssh$
```

Step 7: Export other environment variables:

In your .bashrc file located in $HOME directory, you should add

export BLOBSEER_HOME=$HOME/ release-0.3.3

```
ana@ana-desktop:~$ pwd
/home/ana
ana@ana-desktop:~$ tail .bashrc
    . /etc/bash_completion
fi

#instructions added to bashrc in order to install BlobSeer
#export java home
export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.15/bin
#export library path
export LD_LIBRARY_PATH=$HOME/deploy/lib
#export blobseer home
export BLOBSEER_HOME=$HOME/release-0.3.3
```

Step 8: Check if installation is ok

Locate in the directory where you have installed BlobSeer (in our case release-0.3.3) and run

./scripts/local-deploy.sh

If all the settings are ok, you should obtain the output:

```
ana@ana-desktop:~$ cd release-0.3.3/
ana@ana-desktop:~/release-0.3.3$ ./scripts/local-deploy.sh
WARNING: environment variable CLASSPATH is not set
blobseer-deploy.cfg                          100% 1561     1.5KB/s   00:00
All done!
ana@ana-desktop:~/release-0.3.3$
```

Annex 2: BlobSeer configuration scripts

The BlobSeer configuration scripts can be found in /scripts/blobseer-template.cfg. You can change this configuration. The default one can be seen bellow:

```
# Version manager configuration

vmanager: {

   # The host name of the version manager

   host = ${vmanager};

   # The name of the service (tcp port number) to listen to

   service = "2222";

};


# Provider manager configuration

pmanager: {

   host = ${pmanager};

   service = "1111";

};


# Provider configuration

provider: {

   service = "1235";

   # Maximal number of pages to be cached
```

```
    cacheslots = 1;
    # Update rate: when reaching this number of updates report to provider manager
    urate = 100;
    dbname = "/tmp/blobseer/provider/db/provider.db";
    #dbname = "";
    # Total space available to store pages, in MB (1GB here)
    space = 1024;
    # How often (in secs) to sync stored pages
    sync = 100;
    # How many times to retry writing from client side before giving up
    retry = 1;
    # Use compression?
    compression = true;
};

# Built in DHT service configuration
sdht: {
    # Maximal number of hash values to be cached
    cacheslots = 1000000;
    # No persistency: just store in RAM
    dbname = "";
    # Total space available to store hash values, in MB (128MB here)
    space = 128;
    # How often (in secs) to sync stored hash values
    sync = 10;
```

```
    # Use compression?
    compression = false;
};

# Client side DHT access interface configuration
dht: {
    # The service name of the DHT service (currently tcp port number the provider listens to)
    service = "1234";
    # List of machines running the builtin dht (sdht)
    gateways = (
        ${gateways}
    );
    # How many replicas to store for each metadata entry
    replication = 1;
    # How many seconds to wait for response
    timeout = 10;
    # How big the client's cache for dht entries is
    cachesize = 1048576;
    retry = 1;
};
```

This script also creates blobseer-deploy.cfg file, which is copied to /tmp/blobseer.cfg.

If you modify this file, you have to run in the command line:

```
sh local-deploy.sh blobseer-template.cfg
```

Annex 3: BlobSeer operations

Create a blob

The files used to create, read and write blobs are located in test/ directory. If you want to create a blob which has the size of 64KB, you run the command:

./create_blob test.cfg 65536 1

```
ana@ubuntu:~/release-0.3.4/test$ ./create_blob test.cfg 65536 1
Blob created successfully.
End of test
ana@ubuntu:~/release-0.3.4/test$ vim test.cfg
ana@ubuntu:~/release-0.3.4/test$
```

To write a blob, you run the command:

./test W 1 test.cfg

where:

W – is the write operation
1 – is the blob number
test.cfg - is the configuration file test

To read a blob you run the command:

./test R 1 test.cfg

where:

R – is the read operation
1 – is the blob number
test.cfg - is the configuration file test

Annex 4: MonALISA installation

To compute the trust level of a client, we need to monitor his activity using a monitoring application. As I said above, we will use MonALISA service. This can be downloaded from [9]. After you download the service, you will run the installation script, which is install.sh. This script will create also a farm on your computer, which will be monitored by the service. During the installation, you have to complete the information about your farm (location, name, contact details etc). It is recommended to write real information, because it is easier to locate your farm on the map.

After the MonALISA is installed, you may start to monitor your system, using the command: CMD/START_SER start. All logs from your client activity are stored in ML0.log from the directory where the farm is created.

All the applications needed by the trust module are installed and now we can start to implement this package.

Annex 5: Database content – table policy_events:

event_type character varying(	importance character(1)	description character varying(250)	event_value integer
read data	l	read data	5
append data	m	append data	8
write data	m	write data to a blob	8
policy violation	l	large number of versions for the same data	16
policy violation	l	client without activity for a long period of time	3
loging	l	loging in the system	3
register	l	register in the system	4
deploy	m	deploy a service	6
file versioning	m	see file versioning	5
file versioning	m	access older file versioning	6
delegate rights	h	delegate rights	12
policy violation	h	illegal user	10
policy violation	h	publish no write	12
policy violation	h	write no publish	14
policy violation	m	crowling	13
policy violation	h	write limit exceded	13
policy violation	h	read pages without requesting them from metadata providers	13
policy violation	h	publish different number of pages that those that have been written	15

Annex 6: Input and output files examples

- clients activities [clients_activiy.log file]:

client11,illegal user,2010-06-28
client5,write no publish,2010-06-26
client4,illegal user,2010-06-02
client12,append data,2010-06-11
client5,read pages without requesting them from metadata providers,2010-06-11
client8,delegate rights,2010-06-25
client12,delegate rights,2010-06-17
client13,crowling,2010-06-08
client7,write limit exceded,2010-06-13
client1,delegate rights,2010-06-18
client6,crowling,2010-06-21
client2,client without activity for a long period of time,2010-06-15
client6,large number of versions for the same data,2010-06-23
client8,access older file versioning,2010-06-03
client7,read data,2010-06-11
client9,register in the system,2010-06-29
client10,write no publish,2010-06-19
client12,write data to a blob,2010-06-23
client6,write limit exceded,2010-06-01
client3,illegal user,2010-06-18

- clients allowed actions in the system [clients_actions.log file]:

2010-07-06 : client11 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client5 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client4 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client12 , 2.4825824679710863E-8 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client8 , 1.93365473960193E-8 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client13 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client7 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client1 , 9.335339018615438E-9 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client6 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client2 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client9 , 1.1579375833299138E-8 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client10 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;
2010-07-06 : client3 , 0.0 , 2010-07-06, posible actions: register; login rights; read data; write data; append data;

- clients trust [trust_clients.txt file]:

```
client11,0.0,2010-07-06
client5,0.0,2010-07-06
client4,0.0,2010-07-06
client12,2.4825824679710863E-8,2010-07-06
client8,1.93365473960193E-8,2010-07-06
client13,0.0,2010-07-06
client7,0.0,2010-07-06
client1,9.335339018615438E-9,2010-07-06
client6,0.0,2010-07-06
client2,0.0,2010-07-06
client9,1.1579375833299138E-8,2010-07-06
client10,0.0,2010-07-06
client3,0.0,2010-07-06
```