New Approach for Intrusion Detection in Big Data as a Service in the Cloud

Dounya Kassimi¹, Okba Kazar², Omar Boussaid³, Abdelhak Merizig⁴

^{1,2,4} Smart Computer Sciences Laboratory, Laboratoire d'INformatique Intelligente (LINFI)

Computer Sciences Department, University of Biskra, Biskra, Algeria

dounya_kassimi@yahoo.fr

kazarokba@gmail.com

merizig.abdelhak@gmail.com

*Journal of Digital
Information Management*

³ ERIC Laboratory - Warehouse, Knowledge Representation and Engineering Department of Psychology of Health Education and Development (PSED), University Lumière Lyon 2 France

Omar.Boussaid@univ-lyon2.fr

ABSTRACT: Nowadays, Big Data has reached every area of our lives because it covers many tasks in different operation. This new technique forces the cloud computing to use it as a layer, for this reason cloud technology embraces it as Big Data as a service (BDAAS). After solving the problem of storing huge volumes of information circulating on the Internet, remains to us how we can protect and ensure that this information are stored without loss or distortion. The aim of this paper is to study the problem of safety in BDAAS, in particular we will cover the problem of Intrusion detection system (IDS). In order to solve the problems tackled in this paper, we have proposed a Self-Learning Autonomous Intrusion Detection system (SLA-IDS) which is based on the architecture of an autonomous system to detect the anomaly data. In this approach, to add the autonomy aspect to the proposed system we have used mobile and situated agents. The implementation of this model has been provided to evaluate our system. The obtained findings show the effectiveness of our proposed model. We validate our proposition using Hadoop as Big Data Platform and CloudSim, machine learning Weka with java to create Model of detection.

Subject Categories and Descriptors

D.4.6 [Security and Protection]; Cryptographic controls: **H.2 [Database Management]; I.2.11 [Distributed Artificial Intelligence]**

General Terms:

Big Data, Cloud Computing, Multi-agent System

Keywords: Big Data as a Service, Multi-agent System, Ids, Autonomic System, Security

Received: 10 March 2018, Revised 9 May 2018, Accepted 26 May 2018

DOI: 10.6025/jdim/2018/16/5/258-270

1. Introduction

The advantages and the solutions proposed with the appearance of the new technologies either Cloud computing or Big Data force the logical and physical structures of mass data storage move towards these technologies. In addition, the solutions offered presented in storage and processing of various operations. However, these solutions do not cover all the previous issues specially the one related to the security problem [1].

The exponential growth in data volume generated in the Internet may contain very valuable resource for hackers, crackers and other cybercriminals. The number of computer attacks has grown in quantity and complexity, making defense an increasingly arduous task. Each

computer that suffers an attack has very limited information on who initiated the attack and its origin. In fact, current systems for intrusion detection and response do not follow the growing number of threats [2]. In this context, the need for a highly effective and quickly reactive security system gains importance. In this manner, the organizations must implement an intrusion detection and prevention system (IDPS) to protect their critical data against various kinds of attacks because anti-virus software and firewalls are not enough to provide full protection for their systems [3].

Intrusion detection systems (IDSs) can be split into three types: A network-based intrusion detection system (NIDS), a host-based intrusion detection system (HIDS), and a hybrid-based intrusion detection system [4]. A HIDS detects malicious activities on a single computer where NIDS identifies intrusions by monitoring multiple hosts and examining network traffic. In NIDS, sensors are located at choke points of the network to perform monitoring, often in the demilitarized zone (DMZ) or on network borders and capture all the network traffic. Hybrid-based IDSs detect intrusions by analyzing application logs, system calls, file-system modifications (password files, binaries, access control lists, and capability databases, etc.) and other host states and activities.

Despite its growing importance, current IDS solutions available have limited response mechanisms. Researches focus is on better intrusion detection techniques, response and effective reaction to threats are still mostly manual and rely on human agents to take effect [2]. To ensure a reactive security that handles the IDS solutions, with agent paradigm, we can add an asset to our solution in order to solve this problem through its characteristics such as: autonomy and reactivity. Moreover, since we are working in a cloud environment which contains many sites distributed around the world, mobile agents could facilitate data collection.

In this paper, we propose a real-time autonomous intrusion response system to provide the best defense possible in a short time. In this manner, this paper is organized as follows section 2 presents the area of our research. Section 3 outlines some related works. In section 4, we present our approach and give a description of the proposed architecture. Section 5 shows the validation of the proposed system. Finally, section 6 concludes this paper.

2. Background

The present section shows a brief description for the main element used in our proposal system which is based on both architecture of the autonomous system (MAPE-K cycle) and Big-Data-as-a service (BDaaS) [5], these architectures are represented in figure 1 and figure 2 respectively.

As we can see from Figure 1 that represents the structure

Figure 1. Overall illustration of the autonomous system [5]

of an autonomic system with its MAPE-K cycle [6], this structure is composed of a set of operations that include: monitoring, analysis, planning and executing modules. All the management of the autonomic component is performed by a meta-management element, which makes decisions based on the knowledge base. Moreover, in this system sensors are responsible for collecting information from the managed element. When we collect this information the sensors send them to the monitors where they are interpreted, pre-processed, aggregated and presented in a higher level of abstraction. After this operation, the analysis phase is executed and planning takes place. At this stage, a work plan is generated, which consists of a set of actions to be performed by the executor. Only the sensors and executors have direct access to the managed element. Through the autonomic management cycle, there may be a need for decision-making, thus there is also a requirement for knowledge base [7].

For service providers, there are multiple ways to address the Big Data market with as-a-Service offerings. These can be roughly categorized by level of abstraction, from infrastructure to analytics Software [8], as shown in Figure 2. According to this figure (figure 2), our proposed system takes place in Data Storage Layer and Compute Layer (MapReduce).

Figure 2. BDAAS Stack by job function (proposed by Google [6])

3. Related Works

In the literature, many works have proposed a solution for the intrusion in a different system. In this section, we are going to present some work related to this issue.

In [9], Balasubraiyane et al. propose a distributed architecture, where several independent small processes operate cooperatively to monitor the target system. Its main advantages are its efficiency, its tolerance to faults, its resistance to degradation and its extensibility.

Sodiya in [10] has proposed an architecture entitled MSAIDS, which provides a methodology where the intrusion is handled in two levels. The first one is the lower detection level (LLD) that has the data agents and processing agents. Where the second one is the upper level of detection (ULD) also known as level of confirmation, which is involved in the separation of the process of intrusion detection.

The work proposed by Abraham et al. presented in [11] is based on a hierarchical architecture with Central Analyzer and Controller (CAC) as a core of the Distributed Intrusion Detection System (DIDS). The CAC consists of a database and web server that allows an interactive query by the network administrator for usual information / analysis of the attack and initiating preventive measures. In addition, CAC also performs attack aggregation, building statistics, identifying attack patterns, and enabling rudimentary incident analysis.

The [12] has proposed a system that integrates data mining algorithms and mobile agent technology for detecting known and unknown attacks in the network. This system employs mobile agent technology for processing information from each host. Indeed, mobile agent-based signature detection allows the detection of known attacks.

In [13] suggests an autonomous manager for intrusion detection which introduces a mechanism for multi-attribute auction. In addition, the proposed architecture has a layer of managed resources covering generically all physical devices such as routers, servers or software applications. These resources should be manageable, observable and adjustable. The state of resources refers to all data (events) that reflect the state of existing resources, including logging and real-time events. Furthermore, this architecture also has an autonomous agent that handles some tasks such as detection engine, optimization strategy, autonomic response, and knowledge base module. Besides, the mentioned architecture has a set of agents responsible of Managed Resource (MR) information capture, preprocessing and redundancy removal before final submission to Autonomic Manager agent (AM agents).

Kholidy et al. [14] describes how to extend the current technology and IDS systems. Their proposal is based on hierarchical IDS, to experimentally detect DDoS, host-

based, network based and masquerade attacks. In addition, Kholidy et al. provide some capabilities for self-resilience preventing illegal security event updates on data storage and avoiding a single point of failure across multiple instances of intrusion detection components.

The work of Vollmer et al. presented in [15] describes a new architecture that uses the concepts of autonomic computing based on SOA and external communication layer to create a network security sensor. This approach simplifies the integration of legacy applications and supports a safe, scalable, self-managed structure.

Sperotto et al. [16] presented an autonomic approach to adjust the parameters of intrusion detection systems based on SSH traffic anomaly. The authors in their work have proposed a procedure which aims to tune system parameters automatically and, to optimize system performance. Moreover, the proposed procedure validates their approach by testing it on a probabilistic-based detection test environment for attack detection on a system running SSH.

Kleber et al. [17] suggests the use of autonomic computing to provide a response to attacks on cloud computing environments. Thus, it is possible to provide self-awareness, self-configuration in the cloud. An architecture that uses the expected utility function for choosing an appropriate response is a statistical model to adjust the answers given in order to provide more results. Furthermore, the work proposes the use of Big Data infrastructure using Hadoop to organize the large volume of data and extract information using the Map-Reduce framework.

Boukhlofet et al. [20, 21, 22] proposed a hybrid approach based on mobile agents for the detection of intrusion (HAMA-IDS). The architecture allows the treatment of information directly to the place where it is available through the use of mobile agents (aglets). Thus, the method is based on the platform Aglets for the creation and distribution of agents. These last ones can move from post to post in order to analyze packages collected by the collector agents. The hybrid analysis is assured by the analyzers and redirectors agents, so that they detect possible attacks.

4. Proposed Architecture

In order to propose a solution for the mentioned issue that consists of the resolution of intrusion detecting problem in BDaaS systems. In our work, we propose a model for autonomic intrusion detection system based on the autonomic loop, commonly referenced as MAPE-K (Monitor, Analyze, Plan, Execute and Knowledge Base). In our proposal to monitor and to analyze the mentioned problem, we have used mobile agents [25] to collect data from network traffic for storage and further analytics. In addition, a distributed storage is used in our proposal, in this work we chose Apache Hadoop as storage engine

	IDS	Cloud	Response	Big Data	Algorithm
AAFID [9]	√	X	√	X	genetic
MSAIDS [10]	√	X	√	X	Previously modified
DSCIDS [11]	√	X	√	X	Software calculation paradigm
MAD-IDS [12]	√	X	√	X	Data mining
Wu [13]	√	X	√	X	Auction
Kholiday[14]	√	√	√	X	Holt- Winters
Vollmer [15]	√	X	√	X	Fuzzy
Sperotto[16]	√	X	X	X	Flor-based
IRAS [17]	√	√	√	√	maximization method
SLA-IDS (our approach)	√	√	√	√	SVM

Table 1. Comparative study between different approaches shows the difference between the previous works according to some parameters

Figure 3. General Architecture of the Proposed System

because its performance, scalability and further capabilities to be extended and suffer Map Reduce jobs [26, 27, 28].

For analysis, planning and execution we present a Multi-agent model based on the expected utility function.

4.1 Architecture Description

In this subsection, we explain the components used in our proposed architecture and their roles. As we have mentioned earlier this architecture proposed to solve the intrusion problem. In addition, this architecture is based on a set of agents some are mobile and others are situated

agents. The next figure 3 depicts the proposed architecture.

The proposed architecture shown in figure 3 composed by a set of components given as follows:

- **Interface Layer:** This layer represents the link between the internal and external world (User and system) through a graphical interface. Moreover, the main task of this unit is to gather the client's service requirements and display the obtained results. In addition, in this interface, we assign an agent in order to facilitate the interaction through the use of a model that checks the given data and normalizes

the requests. This operation makes the treatment of clients' request much easier for the system; then, the interface agent sends it to the mediator layer.

- **Mediator Layer:** This layer creates mobile agents for each customer, and this mobile agent is responsible for extracting the required information of each package, so the analyses layer could detect the anomalies. Despite of considering privilege and security reasons in the cloud, we suppose that these agents have the rights to work in data centers.

- **Supervising Layer:** In this layer we supervise the action of each mobile agent, and also we supervise the work of the analyses layer, so we can eliminate the agents that have finished their work and if there is a problem with the mediator layer will assign another agent to continue its work.

- **Analyses Layer:** The job of this layer is to detect the anomaly packages that come from mobile agent created with mediator layer. Moreover, this layer ensures the detection task using a valid model which is in our proposition "SVM". After that, it sends the decision to the next layer (planning layer).

- **Planning Layer:** This layer determines the fate of the packages coming from the mobile agent. After receiving the decision from the analyses layer, this layer checks the protocol used in sending those packages and sends its decision to the executor layer.

- **Executor Layer:** This layer implements the decisions that are taken by the planning layer, by creating a mobile agent to apply the decision, those agents control the mobile

agent created by mediator layer to either allow the storage of the data or delete it.

4.2 The Proposed Intrusion Detection System

After describing the components of our proposed approach, the figure 4 represents the location of our proposed system SLA-IDS (Self-Learning Autonomous Intrusion Detection) in Big data as a service Layer (BDaaS).

4.2.1 Monitoring Layer

The first phase of the MAPE-K autonomic cycle corresponds to monitoring. In this step, sensors are used to obtain data from the users, in our case we consider mobile agents as sensors, so our monitor has a host platform JADE to create mobile agents. Moreover, a Big database to store all the information about the packages that are obtained by the agents from the users to analyze these data. In order to achieve this operation, we have used Hadoop as big data platform to store all these information permanently because its performance, scalability and further capabilities to be extended and suffer Map Reduce jobs, as presented in figure 5.

4.2.2 Analysis Layer

In this step, we generate the model of intrusion detection, as we can see in figure 6. The model used in this work is SVM [24] which is based on the following steps below:

- **Training Step:** In this step, we have used train database from the literature named (NLS_KDD/ KDDTrain) [17]. The used model contains information about the packet (protocol type, service, host login, guest login.....) as features where the labels represented by "numeric, nominal".

Figure 4. BDaaS Market Overview [18]

Figure 5. Architecture Illustration of the Monitoring Phase

Figure 6. The different Steps of Model Creation

- **Testing Step:** After the model is performed with the training step, we test it with another database (KDDTest).

- **Validation Step:** In this step, we validate our model by testing it with another database (KDDTest-21) in order to reduce the error.

4.2.3 Planning and Executor

The second phase is Deployment Model (see Figure 7), we integrate our model in the system, which has an analyzing agent that is capable of detecting the intrusion and leave the decision to the mediator agent in the planning and execution phases.

In the planning phase, we will use an agent called mediator agent, this agent makes the decision of the action after receiving the results from analyzing agent, taking in to consideration the protocol of sending these packets. In case that the used protocol for the packet is TCP, in the event that this protocol detects to an anomaly then it will delete the packet, consequently, it will ask for sending the packet again. Otherwise, the packet will be deleted

without additional actions, and the mobile agent created by the mediator agent will perform this operation. These agents represent the executor phase.

4.3 The Internal Architecture of the used Agents

As we have mentioned in the previous section that we have several agents in this subsection, we are going to present the modules of each used agent.

The concrete architecture of the supervising agent (figure 8) presented in the above figure consists of the following modules:

- **Communication Module:** This module provides all the mechanisms of interaction of the agent with the other agents but also with the module of decision-making (Task Module).

- **Task Module:** It is the brain of the agent. This module is responsible for choosing the appropriate task with the current state of the environment, and those tasks delete single information or delete all the data to store now ones.

Figure 7. Architecture Illustration of the Execution Phase

Figure 8. Concrete Architecture of the Supervising Agent

Figure 9. Concrete Architecture of the Planning Agent

Figure 10. Sequence Diagram of SLA-IDS

The planning agent presented in figure 9 is used in our architecture, it is composed of the next modules:

- **Communication Module:** This module provides all the mechanisms of interaction of the agent with the other agents and with the action module to decide on the number of agents need to be created and then assigns tasks to each one of them.
- **Action Module:** In this module, the planning agent takes the decision to delete the information permanently. In addition, this module offers communication with the customer to re-send the decision, in the event of information

anomaly which depends on the used protocol by the customer, or stores that information if they are sound.

- **Create Agent Module:** Allows planning agent to create a mobile agent to each packet received from the customer.
- **Environment:** This module indicates to the customer, another agent or big data Platform.

In order to resume the scenario used in our proposal, we are going to use the next figure (Figure. 10) which represents the different operations using an UML sequence diagram given as below.

The SLA-IDS checks each transmitted package in order to store by the customers of the cloud computing. First thing, the customers send their requests via the Interface Layer, where these requests will be treated and sent to the Mediator Layer, this one creates for each user a mobile agent so he can supervise his actions, and then it sends this number to Mediator Layer. The Mobile agent converts the packets to Johnson file (), so we can analyze its information and detect the anomaly, then we store them in Hadoop Big data platform permanently to be analyzed and corrected by the Analyzing and Planning Layers.

After Johnson files are stored by mobile agent, we have three analyzing agents (our proposed model "SVM") to analyze the data node in Hadoop. Each agent sends the calculated result to Planning Layer (Mediator Agent) to check the protocol that is used to send the anomaly packets in this situation we have two cases. The first case, if it is a TCP protocol the packet will be deleted and the system will ask the customers to resend these packets. The second case, if it is not a TCP then these packets will be deleted without additional action.

The decision of the mediator agent will be executed in the Executor Layer by a mobile agent that is created in this Layer each agent is responsible for one packet at a time. When the delay of the anomaly packet is complete this packet will be stored and the Jonson files will be deleted.

5. Validation

To evaluate the performance of the proposed system, we use a Windows operating system machine with 4G RAM and 2.6 GHT micro-processor. In our experiment, we have used several Big Data bases downloaded from [29, 30, 31, 32 and 33] to construct our model. Furthermore, we used five sites each one of them contains one heterogeneous big data base. In addition, since we have used Multi-agent system in our solution we used JADE platform to implement the system components.

In this section we are going to represent the same algorithms that we used for the validation of our proposal architecture, starting with the interface of the application (Figure 11).

Figure 11. Upload Data Base

A message dialog appears when the upload is:

In case of error, such as the Upload URL is mistyped, an error message is displayed as follows:

The UploadServlet class to read upload data and save it as a file on disk. We implement the code in the servlet's doPost() method.

5.1 Checking if the Request having Upload Data

```
// checks if the request actually contains upload file
if (!ServletFileUpload.isMultipartContent(request)) {
    PrintWriter writer = response.getWriter();
    writer.println("Request does not contain upload data");
    writer.flush();
    return;
}
```

It checks if the request contains upload data (indicated by the attribute enctype="multipart/form-data" of the upload form). If not, we print a message to the user and exit the method.

5.2 Configuring Upload Settings

Next, we configure some settings using the constants defined previously.

The DiskFileItemFactory class manages file content either in memory or on disk. We call setSizeThreshold() method to specify the threshold above which files are stored on disk, under the directory specified by the method setRepository(). In the above code, we specify the temporary directory available for Java programs. Files are

```
// configures upload settings
DiskFileItemFactory factory = new DiskFileItemFactory();
factory.setSizeThreshold(THRESHOLD_SIZE);
factory.setRepository(new File(System.getProperty("java.io.tmpdir")));

ServletFileUpload upload = new ServletFileUpload(factory);
upload.setFileSizeMax(MAX_FILE_SIZE);
upload.setSizeMax(MAX_REQUEST_SIZE);
```

stored temporarily in that directory and they will be deleted. So we need to copy the files from the temporary directory to the desired location.

The ServletFileUpload is the main class that handles file upload by parsing the request and returning a list of file items. It is also used to configure the maximum size of a single file upload and maximum size of the request, by the methods setFileSizeMax() and setSizeMax() respectively.

5.3 Creating Directory to Store Upload File

Because upload files are stored temporarily under a temporary directory, we need to save them under another permanent directory. The following code creates a directory specified by the constant **UPLOAD_DIRECTORY** under web application's root directory, if it does not exist.

```
// constructs the directory path to store upload file
String uploadPath = getServletContext().getRealPath("")
    + File.separator + UPLOAD_DIRECTORY;
// creates the directory if it does not exist
File uploadDir = new File(uploadPath);
if (!uploadDir.exists()) {
    uploadDir.mkdir();
}
```

5.4 Reading Upload Data and Save to File

The code of parsing the request to save upload data to a permanent file: The parseRequest() method returns a list of form items which will be iterated to identify the item (represented by a FileItem object) which contains file upload data.

The method isFormField() of FileItem (Figure 12) class checks if an item is a form field or not. A non-form field item may contain upload data, thus the check:

```
if (!item.isFormField()) {
}
```

Using KDD data ((Figure 13) it contain 42 attributes) to create the model of detection we used weka classifier from java and we got the result below (Figure 14 and Figure 15):

```
List formItems = upload.parseRequest(request);
Iterator iter = formItems.iterator();

// iterates over form's fields
while (iter.hasNext()) {
    FileItem item = (FileItem) iter.next();
    // processes only fields that are not form fields
    if (!item.isFormField()) {
        String fileName = new File(item.getName()).getName();
        String filePath = uploadPath + File.separator + fileName;
        File storeFile = new File(filePath);

        // saves the file on disk
        item.write(storeFile);
    }
}
```

Figure 12. Reading Upload Data and Save to File

KDDTrain+.arff																			
Relation: KDDTrain																			
No.	1: duration	2: protocol_type	3: service	4: flag	5: src_bytes	6: dst_bytes	7: land	8: wrong_fragment	9: urgent	10: hot	11: num_failed_logins	12: logged_in	13: num_compromised	14: root_shell	15: su_attempted	16: num_root	17: num_file_creations	18: num_shells	
	Numeric	Nominal	Nominal	Nominal	Numeric	Numeric	Nominal	Numeric	Numeric	Numeric	Nominal	Nominal	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric
1	0.0	tcp	ftp_data	SF	491.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	udp	other	SF	146.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	tcp	private	SO	0.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	tcp	http	SF	232.0	8153.0	0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	tcp	http	SF	199.0	420.0	0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	tcp	private	REJ	0.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	tcp	private	SO	0.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	tcp	private	SO	0.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	tcp	remot...	SO	0.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	tcp	private	SO	0.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	tcp	private	REJ	0.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	tcp	private	SO	0.0	0.0	0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	tcp	http	SF	287.0	2251.0	0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	tcp	ftp_data	SF	334.0	0.0	0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0

KDDTrain+.arff																			
Relation: KDDTrain																			
32: dst_host_count	33: dst_host_srv_count	34: dst_host_same_srv_rate	35: dst_host_diff_srv_rate	36: dst_host_same_srv_port_rate	37: dst_host_srv_diff_host_rate	38: dst_host_error_rate	39: dst_host_srv_error_rate	40: dst_host_error_rate	41: dst_host_srv_error_rate	42: class									
Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Nominal									
150.0	25.0	0.17	0.03	0.17	0.0	0.0	0.0	0.0	0.05	0.0 normal									
255.0	1.0	0.0	0.6	0.88	0.0	0.0	0.0	0.0	0.0	0.0 normal									
255.0	26.0	0.1	0.05	0.0	1.0	0.0	1.0	0.0	0.0	0.0 anom...									
30.0	255.0	1.0	0.0	0.03	0.04	0.03	0.01	0.0	0.01	0.01 normal									
255.0	255.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0 normal									
255.0	19.0	0.07	0.07	0.0	0.0	0.0	0.0	0.0	1.0	1.0 anom...									
255.0	9.0	0.04	0.05	0.0	0.0	1.0	1.0	0.0	0.0	0.0 anom...									
255.0	15.0	0.06	0.07	0.0	0.0	1.0	1.0	0.0	0.0	0.0 anom...									
255.0	23.0	0.09	0.05	0.0	0.0	1.0	1.0	0.0	0.0	0.0 anom...									
255.0	13.0	0.05	0.06	0.0	0.0	1.0	1.0	0.0	0.0	0.0 anom...									
255.0	12.0	0.05	0.07	0.0	0.0	0.0	1.0	0.0	1.0	1.0 anom...									
255.0	13.0	0.05	0.07	0.0	0.0	1.0	1.0	0.0	0.0	0.0 anom...									
8.0	219.0	1.0	0.0	0.12	0.03	0.0	0.0	0.0	0.0	0.0 normal									
2.0	20.0	1.0	0.0	1.0	0.2	0.0	0.0	0.0	0.0	0.0 anom...									
255.0	1.0	0.0	0.07	0.0	0.0	1.0	1.0	0.0	0.0	0.0 anom...									

Figure 13. KDDTrain Data

```

Time taken to build model: 3075.29 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      122704          97.405 %
Incorrectly Classified Instances    3269           2.595 %
Kappa statistic                    0.9478
Mean absolute error                 0.026
Root mean squared error             0.1611
Relative absolute error             5.2149 %
Root relative squared error         32.2954 %
Total Number of Instances          125973

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0,986   0,039   0,967     0,986   0,976     0,948   0,973     0,960   normal
                0,961   0,014   0,983     0,961   0,972     0,948   0,973     0,963   anomaly
Weighted Avg.   0,974   0,028   0,974     0,974   0,974     0,948   0,973     0,961

=== Confusion Matrix ===
      a    b  <-- classified as
66370   973 |      a = normal
 2296 56334 |      b = anomaly

```

Figure 14. Result of KDDTrain

As shown on table 1, our work uses Big Data as a service in the cloud to locate anomaly data and be able to provide a response that takes into consideration the type of protocol that is used in sending this data. The contribution of our research was to provide a system that can detect the anomaly data when the customers of the cloud aim to store their data.

6. Conclusion

The paper suggests the use autonomous computing architecture to detect data anomaly when the customers of the cloud aim to store their information. The work proposes the use of Big Data infrastructure using the Hadoop platform to organize the large volume of data and

Figure 15. Result of KDDTest

multi-agent system to provide intrusion detection. To develop our proposition we integrate the big data as a layer in the cloud environment (Big Data as a service), and we used a mobile agent to guarantee the quick reaction of our system to the anomaly data. That is to provide a self-healing intrusion detection system.

We are thinking as perspective of this work to integrate semantic aspect and apply other IDS approaches, and we are also trying to use PHM (Prognostics and health management) methods to apply a data mining for Medical Big Data.

References

- [1] Bosworth, S., Kabay, M. E., Whyne, E. (2014). COMPUTER SECURITY HANDBOOK, Published by John Wiley & Sons, Inc., Hoboken, New Jersey, (2014) p, 111-115.
- [2] Lumpur, K. (2010). An investigation and survey of response options for Intrusion Response Systems (IRSS).
- [3] Sandhu, U. A., Haider, S., Naseer, S., Ateeb, O. U. (2011). A survey of intrusion detection & prevention techniques. In: 2011 *International Conference on Information Communication and Management*, IPCSIT (Vol. 16). 66-71.
- [4] Beigh, B. M., Peer, M. A. (2012). Intrusion Detection and Prevention System: Classification and Quick Review, *ARPN Journal of Science and Technology*, 2 (7) 661-675.
- [5] Rutten, Eric., Marchand, Nicolas., Simon, Daniel. (2015). Feedback Control as MAPE-K loop in Autonomic Computing. [Research Report] RR-8827, INRIA Sophia Antipolis - Méditerranée; INRIA Grenoble -Rhône-Alpes.
- [6] <https://fr.slideshare.net/BDaaSmeetup/bdaas-meetup-the-emergence-of-bigdataasaservice>
- [7] Huebscher, M. C., McCann, J. A. (2008). A survey of autonomic computing degrees, models, and applications, *ACM Computing Surveys (CSUR)*, 40 (3) 7.
- [8] Hariri, S., Khargharia, B., Chen, H., Yang, J., Zhang, Y., Parashar, M., Liu, H. (2006). The autonomic computing paradigm, *Cluster Computing*, 9(1) p. 5–17.
- [9] BIG DATA-AS-A-SERVICE: A Market and Technology Perspective', White Paper, EMC July 2012.
- [10] Balasubramanian, S., Omar, GFJ., David, I., Eugene, S., Diego, Z. (1998). AAFID – Autonomous Agents For Intrusion Detection. Technical report 98/05, COAST Laboratory Purdue University, (June).
- [11] Sodiya, A. S. (2006). Multi-level and secured agent-based intrusion detection system. *Journal of Computing and Information Technology*, 14 (3) 217-223.
- [12] Abraham, A., Jain, R., Thomas, J., Han, S. Y. (2007). D-SCIDS: Distributed soft computing intrusion detection system. *Journal of Network and Computer Application*, 30, 81-98.
- [13] Brahmi, I., Ben Yahia, S., Poncelet, P. (2010). MAD-IDS: Novel Intrusion Detection System using Mobile Agents and Data Mining Approaches. *Lecture Notes in Computer Science*, Volume 6122, p. 73-76.
- [14] Wu, Q., Zhang, X., Zheng, R., Zhang, M. (2013). An Autonomic Intrusion Detection Model with Multi-Attribute Auction Mechanism, 10 (1) 56–61.
- [15] Kholidy, H., Erradi, A., Abdelwahed, S., Baiardi, F. (2013). A hierarchical, autonomous, and forecasting cloud IDS, p. 213–220.
- [16] Vollmer, D., Manic, M., Linda, O. (2013). Autonomic Intelligent Cyber Sensor to Support Industrial Control Network Awareness, *IEEE Transactions on Industrial Informatics*, vol. p (99) 1–1.
- [17] Idss, A.-b., Case, S. S. H., Sperotto, A., Mandjes, M., Sadre, R., Boer, P.- t. D., Pras, A., de Boer, P.-T. (2012). Autonomic Parameter Tuning of Anomaly-Based IDSs: an SSH Case Study, *IEEE Transactions on Network and Service Management*, vol. 9, p. 128–141, (June).
- [18] Big data et Cloud computing : le pari gagnant des offres BDaaS <https://www.riskinsight-wavestone.com/2015/08/big-data-et-cloud-computing-le-pari-gagnant-des-offres-bdaas/>
- [19] <https://www.riskinsight-wavestone.com/2015/08/big-data-et-cloud-computing-le-pari-gagnant-des-offres-bdaas/>

- [20] Boukhlouf, D., Kazar, O. (2012). Hybrid Approach based Mobile Agent for Intrusion Detection System: HAMA-IDS. *Journal of Information Security Research*, Volume 3, p. 30-41 (March).
- [21] Boukhlouf, D., Kazar, O. (2012). Intrusion Detection System: Hybrid Approach based Mobile Agent. ICEELI 2012: International Conference on Education and E-Learning Innovations July 1-3 Sousse, Tunisia.
- [22] Boukhlouf, D., Kazar, O. (2012). Hybrid Approach based Mobile Agent for Distributed Intrusion Detection System. *In: Proceeding ICDS 2012: International Conference on Distributed Systems and Decision*, ISSN 2335-1012, Oran Algeria 2012.
- [23] https://github.com/defcom17/NSL_KDD.git
- [24] Merizig, A., Kazar, O., Sanchez, M. L. (2018). A Dynamic and Adaptable Service Composition Architecture in the Cloud Based on a Multi-Agent System. *International Journal of Information Technology and Web Engineering (IJITWE)*, 13(1) 50-68.
- [25] Frécon, L., Kazar, O. (2009). manuel d'intelligence artificielle, Roman Polytechnic And University Presses.
- [26] Saouli, H., Kazar, O., KASSIMI, D. (2016). Applications et enjeux des Big Data dans le contexte des défis mondiaux. *In: Proceedings 10th of Les Avancées des Systèmes Décisionnels (ASD)*, Annaba, Algérie (2016) 14-16 May.
- [27] Kassimi, D., Kazar, O., Saouli, H., Boussaid, O., Saifi, S., Hassani, I. (2017). A new approach based mobile agent system for ensuring secure Big Data transmission and storage. *In: International Conference on Mathematics and information Technology*, Adrar, Algeria, December 4 - 5, p. 196-200.
- [28] KASSIMI, D., Kazar, O., Saouli, H., Boussaid, O. (2017). Design and Implementation of a New Approach using Multi-Agent System for Security in Big Data. *International Journal of Software Engineering and its Applications*, September 2017, p.1-14.
- [29] consulted on 10/03/2016: <http://aws.amazon.com/fr/publicdatasets/>
- [30] consulted on 25/03/2016: <https://archive.ics.uci.edu/ml/datasets.htm>
- [31] consulted on 10/03/2016: <http://www.kdnuggets.com/datasets/>
- [32] consulted on 10/03/2016: <https://www.opensciencecloud.org/publicdata/>
- [33] consulted on 10/03/2016: http://en.wikipedia.org/wiki/Wikipedia:Database_download