Malware code detection for Cloud computing – A rule based probabilistic approach

Sanjay Chakraborty
Institute of Engineering & Management, India
schakraborty770@gmail.com

Lopamudra Dey
Heritage Institute of Technology, India
lopamudra.dey1@gmail.com

Abstract: *Security is one of the major concerns in cloud computing. Security is obtained to prevent threats that affect both the cloud user and cloud provider. Malicious code deployment is the main cause of threat.This paper mainly proposes a novel malicious code detection technique which provides security in the traditional cloud computing. Advanced malware detection system (AMDS), aimed at guaranteeing increased security to cloud computing. AMDS can be deployed on a VMM which remains fully transparent to guest VM and to cloud users. AMDS prevents the malicious code running in one VM (infected VM) to spread into another non-infected VM with help of hosted VMM.The main aim of this paper is to detect the malicious code by some advanced technique and warn the other guest VMs about it.A prototype of AMDS is partially implemented on one popular open source cloud architecture – Eucalyptus. It creates less overhead as well as occupies less storage space compare to other well-known anti-viruses.It is also less expensive to implement compare to others. In this paper,AMDS technique follows a rule based probabilistic approach to detect the malicious code among several codes. At last a comparison analysis of this proposed approach is done with other traditional approaches.*

Keywords: *Comprehensibility, Complexity, Malware, Memory utilization, probabilistic, Processing overhead, Security.*

I. INTRODUCTION

Cloud computing paradigm is responsible for today’s high speed shared internet. It renders the internet as a large repository where resources are virtualized and used to provide services to the cloud users. Cloud represents the internet as a large entity of resources with all the pros and cons of this pervasive system. It brings revolution in the field of internet. Scientific and engineering applications, data mining, computational financing, gaming and social networking, as well as many other computational and data-intensive activities can benefit from cloud computing [9].Cloud users not only achieve the benefits of cloud services but also used to face several challenges in terms of security.
Traditional cloud architecture is totally based on Virtual machine monitor (VMM) running on host OS and multiple Virtual machine (VM) instances running on guest OS. So the common security challenge faced by cloud users is the deployment of malicious code in one VM which affects other VMs.

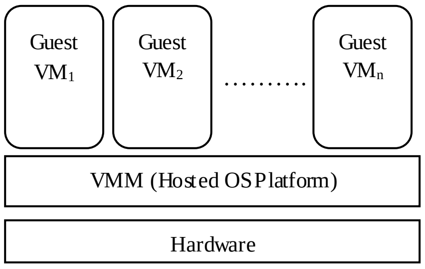


Fig.1. Traditional VM architecture

This paper mainly examines and detects those vulnerable codes by using a new probabilistic approach and informs the other non-infected hosted machines in terms of warning.

Beside this, other privileged instructions are totally controlled by VMM of the system. This paper also analyses different cloud architectures and models and also develops an architecture framework, named “Advanced malware detection system (AMDS)” which provides malware detection service through VMM in fully transparent mode. It actually proposes this AMDS as a prototype and tries to implement and deploy on one popular open source cloud architecture – Eucalyptus partially.Clearly, there is a very urgent need to find, not just a suitable method to detect infected files, but too build a smart methodology that can detect new viruses by studying the structure of the program made by malware.
This paper is organized as following. Some prior works on cloud security through malware detection & virtualization are analyzed and discussed in section II.

II. RELATED WORK

Cloud security plays a very important role in cloud computing architecture. Several works have been done to handle security issues in cloud architecture. This paper is mainly based on the concept of introducing a new malicious code detection system which is applied on a famous Eucalyptus cloud architecture.
One paper has given a brief descriptions about different security challenges and solutions in cloud computing. This paper mainly discussed about policy, software and hardware security[13][14]. A short paper has discussed a highly scalable security solution for guest VMs. In this work, they have described how they securely bridged the semantic gap into the operating system semantics. The presented solution enables novel security services for fast changing cloud environments where customers run a variety of guest

operating systems, which need to be monitored closely and quarantined promptly in case of compromise [2] .The basic idea regarding next generation cloud architecture can be obtained from a paper [11].
There are several techniques by which a malware code can be detected. Those various techniques have been discussed by some papers [10][7]. The aim of this paper has been influenced by a secure virtualization technique known as advanced cloud protection system (ACPS) which provides fully transparent security to cloud resources applicable on Eucalyptus and Open-ECP architectures [4].A novel malware detection technique is also introduced as a standard to detect malicious codes deployment [1] [7].
So a fast changing internet faces different challenges in the form of malwares, viruses etc. and various techniques has been introduced to detect and prevent those challenges.All these new relevant features, as well as extensive experiments on both security and performance, make the present proposal a novel contribution [8].
Some solutions are given to prevent malware injection attack in cloud through storing OS type of the customer virtually. As cloud OS is platform independent, before launching the infected instances or services on the actual OS of customer's VM, it should be cross checking on this virtual OS type [17]. However, when a customer uses a cloud system, the image of customer's VM is integrated with the hardware level of the system, because it is very difficult for an attacker to intrude in the IaaS level. A file allocation table (FAT) with the Hypervisior is utilized to check over the previous instances that had been already executed from the customers machine can be put to determine the validity and integrity of the new instance [17].
An approach based on a dedicated cloud resilience manager (CRM) which is installed in host VM. This CRM software component consists of a Network Analysis Engine (NAE), and a System Analysis Engine (SAE) for detecting infected packets and a System Resilience Engine (SRE) and Coordination and Organisation Engine (COE) for destroying an infected VM or blocking information destined to a vulnerable VM. This system is specially built for preventing Kelihos injection. This approach is also useful for gathering and analysing data at the network level through the use of network monitors [18].
One approach has been recently introduced where feature based malware detection can be done using one class support vector machine (SVM) formulation at the hypervisor level. These features are gathered at the system and network levels of the cloud node. This security system is built by considering not only system-level data, but also network-level data depending on the attack type with no prior-knowledge. This approach is flexible and useful for real-time data where the behaviour or features of the data are continuously changing [19].

III. PROPOSED APPROACH

A cloud is a pool of virtualized resources which are used to fulfill the users requirements based on 'pay-as-you-go' approach.Within the cloud paradigm, concepts such as virtualization, distributed computing and utility computing are applied. As everyone knows cloud service layer is divided into three parts, SaaS (Software as a service), PaaS (Platform as a service) and IaaS (Infrastructure as a service). The main focus of this paper is to be given on the lowest layer(IaaS) of the model where AMDS system is implemented and working (Fig.2).
The entire approach is basically probabilistic approach which based on three parameters, such as, Memory utilization (MU), Processing overhead (PO) and Comprehensibility [7]. These are the main factors which are used to detect the vulnerability of a malicious code. These factors are also called malware signatures. These parameters are related with the behaviour of a code during its execution. They use to maintain some threshold values to distinguish between normal and abnormal code execution.

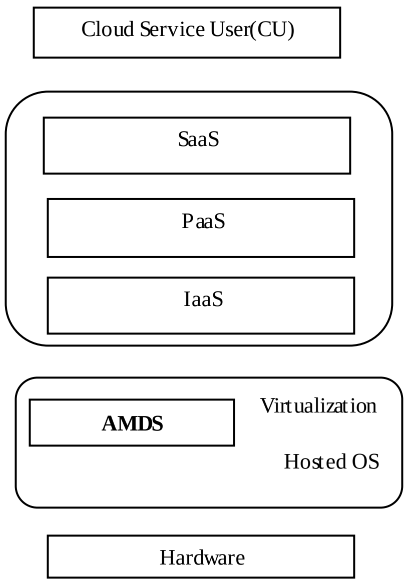


Fig.2. AMDS system's Location in Cloud service model

A. *Memory utilization (MU):*

Cloud service based memory utilization which is an effective mechanism for allocating memories in cloud computing environment [15].Memory utilization is one of the important factors by which the performance of an executing code can be measured. Memory utilization means how much memory it is using when a request process is successfully executing. The complex code needs more memory compare to a simple code. It is also being represented in terms of space complexity. It is a common fact that unnecessary memory utilization is not desirable in cloud computing services. This total approach is very similar based with the human analysis factors for malware code detection. Memory utilization factor is evaluated here based on the usage of kernel and physical memory (service based resources) of the main cloud server (on which VMM running) and represents the difference of the occupied space (in percentage) between normal and malicious code. The proposed AMDS plays the role to measure the memory utilization of those end user service requests to make a primary guess of a malicious code (by the help of some threshold value checking) [10].

B. *Processing Overhead (PO):*

In computer science, overhead is any combination of excess or indirect computation time, memory, bandwidth, or other resources that are required to attain a particular goal.In this part this factor mainly deals with the processor overhead of the main cloud server system. Processor overhead measures the amount of work a computer's central processing unit can perform and the percentage of that capacity that's used by individual computing tasks. Some of these tasks involve supporting the functions of the computer's operating system. These include communication with internal components, such as hard drives and graphics processing units, with connective functionality such as networking support, and with external components like your computer's monitor. A malicious code or a software bug can draw too many CPU clock cycles and overtax your system unexpectedly [6].The proposed AMDS plays the role to measure the processor overhead of those end user service requests to make a primary guess of a malicious code (by the help of some threshold value checking).

C. *Comprehensibility (C):*

Comprehensibility in malware detection is how easy to read and understand the instructions within the executing code. This means it points to the complexity of the code from reading and understanding.

Comprehensibility is measured here by using the popular concept of Complexity Profile Graph (CPG).Complexity Profile Graph is a graphical representation to measure statement level complexity in a code. The CPG divides the entire source code into some measurable units which is called CPG segments.CPG segments consist of simple statements(e.g., assignment, procedure call) and clauses (e.g., declarations) in the underlying source language [1] [12].Based on the detailed analysis of statement level complexity of a source program, it can be divided into two types of statements. They are listed in Table I.

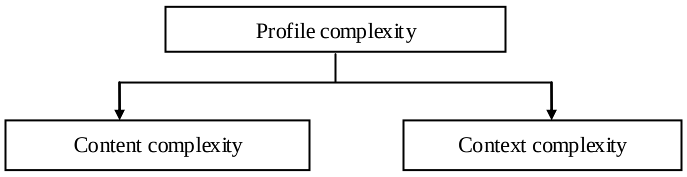
CPG is based on a profile metric which is actually represents different granular level complexity. The profile metric can be measured as,

$$\boldsymbol{F = \{\mu_x \mid X \text{ is a measurable unit}\}}$$

Here **X** is a different CPG segments and ***F*** calculates a specific aspect of ***X***'s complexity. Again the profile complexity is further divided into two basic complexities,

TABLE I
STATEMENT-LEVEL CLASSIFICATION

Simple statement	**Compound Statement**
assignment statement	• IF condition THEN • ELSIF condition THEN • ELSE • END IF;
delay statement	CASE expression, END CASE;
exit statement	Loop statements;
goto statement	Block statement: • [Name][DECLARE] • BEGIN • END [Name];
null statement	Record declaration;
procedure call	Generic Procedure
return statement	Generic Function
declarations	



C.I. Content complexity(£(X))

Content complexity is used to measure the amount of information contained within a measurable unit or segment. It does not give any care on quality. As for example, camera_1 of type real camera and camera_2 of type toy camera. Although the camera_1 structure may be much more complicated than camera_2 structure. Hence those two things his treated here as a camera only. It is very similar to semantic analysis of a code. The content complexity and the context complexity both are independent but both are required to measure the overall profile complexity of a source code.Currently, the content contribution is constrained to be between 0 and 3, while the context contribution is constrained to be between 0and 15 [1]. This design provides the effect that the content complexity is a ripple riding on the curve of the context complexity [1].

Although the content complexity for most tokens is indeed held constant, i.e., 1.0, there are few exceptions: left parenthesis, logical operators (e.g., *and*, *or*, *not*), and comparison operators(e.g.,>, <, =). Here left parenthesis increases complexity and the logical operator adds two simple expressions together to deal with more complexity.

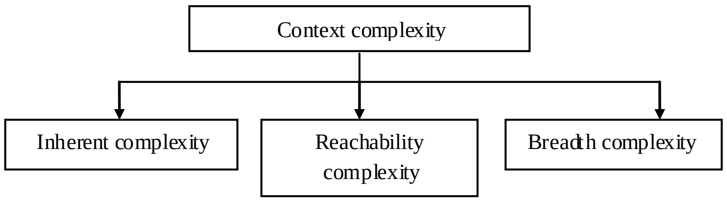
The content complexity, £(X),of a CPG segment **X** is defined as the natural logarithm of the summation of all of its tokens weight contributions [1].

$$\boldsymbol{\text{£}(X) = \ln \sum_{T \epsilon X} \text{Weight } (T)}$$

Where £(X) is ranged from 0 to 3 and Weight (T) should be less than 20 [1].

C.II. Context complexity (¥(X))

The context complexity provides the baseline level of complexity for segments of simple statements nested within a compound statement, which itself may be nested several levels deep. The context complexity of a segment will be the summation of the complexities of all the compound statements in which it resides. This means each compound statement contributes to the overall level of the complexity platform which is uniform for statements within it [1].



The inherent complexity, **I,** is a subjective measure to deal with difficulty nature of a compound statement. The reachability complexity, **R**, indicates the difficulty of reaching a statement with respect to its path predicate [1] [5]. The path predicate means a set of Boolean condition complexity. The Breadth complexity, **B**, represents the amount of computation involved in a compound statement, and is approximated by the number of statements nested within the compound statement.

These three complexities are combined in the following way for a segment X within a compound statement Y.

$$¥(X) = c1 * I(Y) + c2 * R(Y) + c3 * B(Y)$$

with weighting coefficients **c1** = 1.0, **c2** = 1.0, and **c3** = 0.1. Combining the content complexity, **£(X)** and the context complexity, ***¥(X)*** gives the profile metric, µ**(X)**, for a segment. That is,

$$\mu_X = s1 * £(X) + s2 * ¥(X)$$

Where scaling factors, **s1** and **s2**, are set to 1.0 [1].

TABLE II
EXAMPLE OF SEGMENTS AND COMPOUND STATEMENT WEIGHT CHART

Segments	**Weight**	**Compound statement**	**Weight**
Logical operators	1.5	SELECT, ACCEPT	4
Comparison operators	1.5	CASE, IF, ELSEIF	3
Left Parenthesis	1.3	WHILE	3
Identifiers	1.0	FOR,BASIC LOOP, EXIT	2
Others	1.0	BLOCK	1
etc.	etc.	etc.	etc.

D. *Rule Based Probabilistic Approach:*

Based on the discussion and understanding of the above three factors a malicious code can be detected with the help of some threshold values prediction. With the guess of those threshold values a set of basic rules is defined to detect the nature of the malware code.

If the memory utilization crosses 80% ($>TH_{MU}$) and CPU utilization or Processing overhead crosses 90% ($>TH_{PO}$), then the code will be treated as a probable vulnerable code (comprehensibility is also taken into account of consideration). The practical application on general and malicious codes on WINDOWS environment is analyzed and shown in the result analysis section of this paper. This paper uses a probabilistic boolean rule based approach to represent the malicious code status.

TABLE III
BOOLEAN STATUS REPRESENTATION BY THRESHOLD ANALYSIS

	MU / PO / C	Status
Malicious code	$>TH_{MU}$ / $>TH_{PO}$ / $>TH_C$	1
	$<TH_{MU}$ / $<TH_{PO}$ / $<TH_C$	0

From the above boolean analysis it can be easily said that the three factors are organized into eight different ways. It can be represented by the help of following truth TableIV,

TABLE IV
TRUTHTABLE OF COMPLEXITY FACTORS WITH RULES

Memory Utilizati on (MU)	**Processing Overhead (PO)**	**Comprehensibility (C)**	**Code Vulnerability Status (RULES)**
0	0	0	VERY LOW
0	0	1	LOW
0	1	0	LOW
0	1	1	HIGH
1	0	0	LOW
1	0	1	HIGH
1	1	0	HIGH
1	1	1	VERYHIGH

This approach organizes these three factors according to some priority based rules. Obviously, this priority is entirely dependent on the decision of the system where this approach is implemented.

According to this paper the priority is listed in Table V,

TABLE V
PRIORITY BASED FACTORS CLASSIFICATION

Factors	**Priority**
Processing Overhead (PO)	HIGH
Comprehensibility (C)	MEDIUM
Memory Utilization (MU)	LOW

Here, Memory Utilization (MU) is considered as a low priority factor due to its high availability and low storage cost.

These are the three basic factors which are mainly analyzed in this paper for the detection guess of the malicious codes. Here, AMDS is partially implemented over Eucalyptus architecture. Eucalyptus is composed of: a node controller (NC) that controls the execution, inspection and termination of VM instances on the host where it runs; a Cluster Controller (CC) that gathers information about VM and schedules VM execution on specific node controllers;

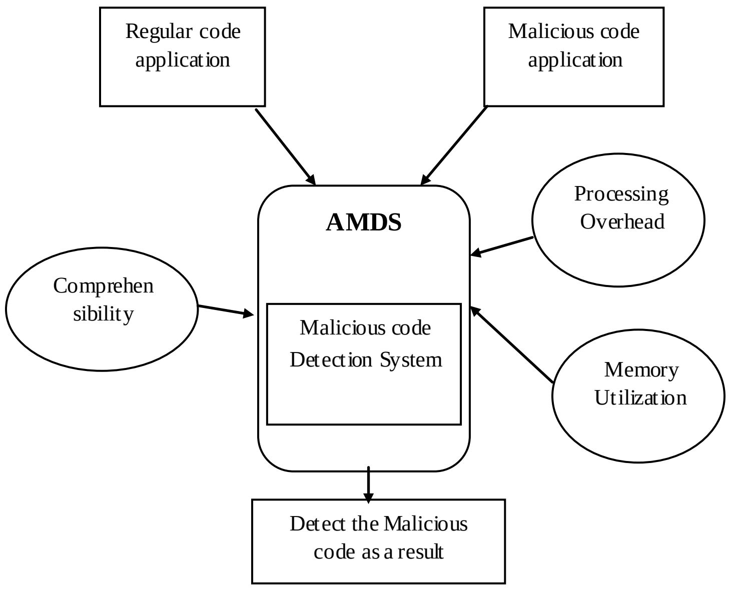

Regular code application
Malicious code application
AMDS
Processing Overhead
Comprehen sibility
Malicious code Detection System
Memory Utilization
Detect the Malicious code as a result

Fig.3. Functionality of AMDS system in Cloud

further, it manages virtual instance networks; a Storage Controller(SC) that is, a storage service providing a mechanism for storing and accessing VM images and user data. A more detailed description of how AMDS integrates with the Eucalyptus component is reported in Fig.4. On Eucalyptus the AMDS can be deployed with the Cloud Controller, the Cluster Controller and, most importantly, the Node Controller.

IV. RESULT ANALYSIS

This section shows how this proposal copes with attacks the cloud can be subject to in real environments. In particular, here the practical experiments performed to assess the resilience of the proposed system and also provide discussion about the fitness of the proposal into the Eucalyptus architecture. The detection capabilities of this AMDS system are assessed against known attack techniques. However, since source code for many attacks is not publicly available, it performed this test by simulating the attack steps.
Here the AMDS system is partially implemented in the Eucalyptus architecture due to the architectural (infrastructure) limitation. That's why the result is solely analysed on the basic Window 7 (64 bit) OS, x86 micro architecture, 2.89 GHz processor and 2GB RAM. Here the load factors of the memory as well as the processor are mainly tested by taking threshold values in account. The AMDS system is mainly used to detect the malicious code based on the above discussed symptoms and stores them as a set of warnings in a dedicated thread storage pool (as per Fig. 4). The limited size of this pool is provided by host-site database. Based on the warnings the Guest VM or Guest OS will decide to accept that code or not. One simple example is taken here,

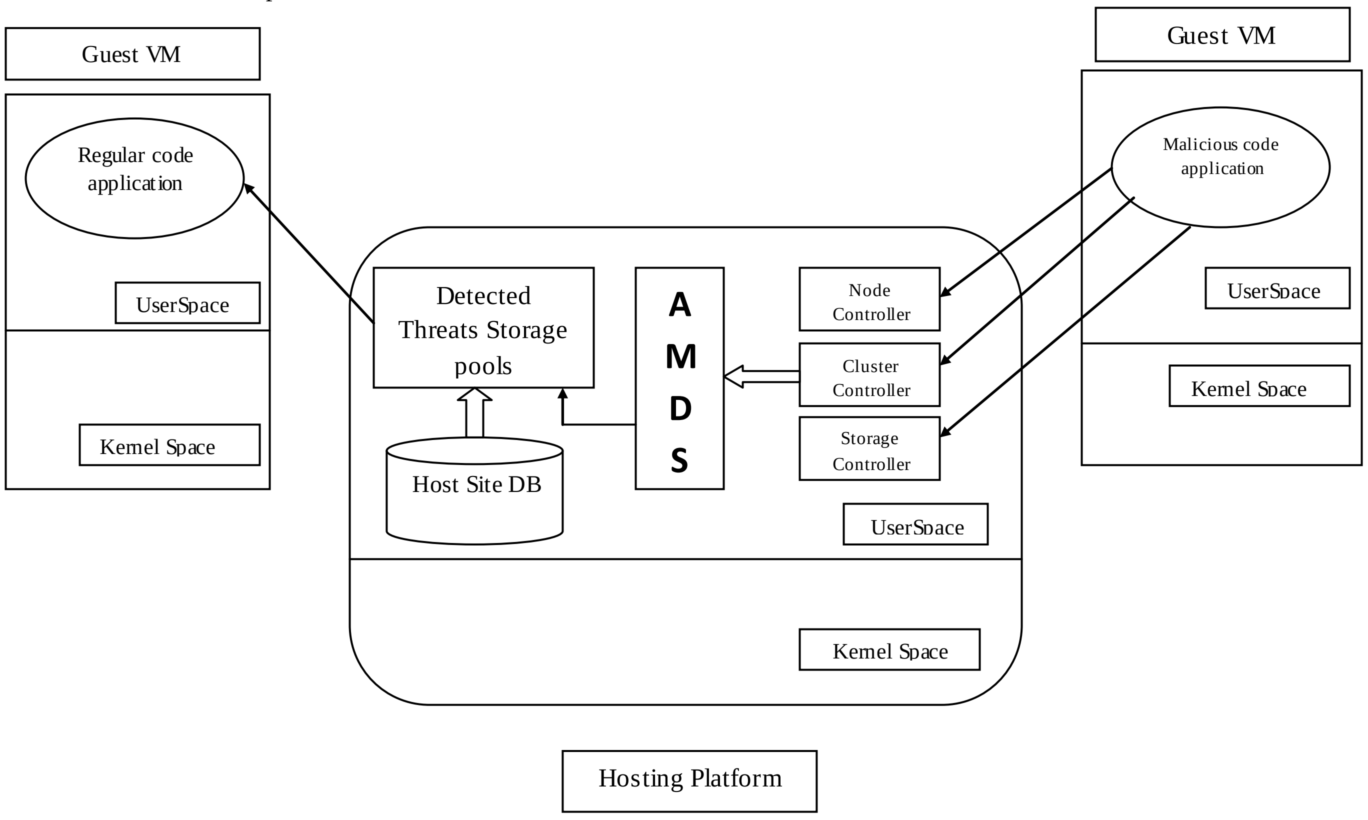


Fig.4. Combine AMDS system with Eucalyptus-Architecture

TABLE VI
Virus Code vs. Regular Code

Virus Code 1	Regular Code 1
#include<stdio.h>	#include<stdio.h>
#include<stdlib.h>	void main()
void main()	{
{	i=0;
while(1)	while(i<3)
{	{
system("dir>>â•ša.exe");	printf("Hello");
}	i++;
}	}
	}

Virus code:
Number of tokens are Weight (T) =5 (Simple statement like assignment, goto, null etc. has only one segment and compound statement like if, CASE, loop, block contain multiple segments).
So, Content analysed metric $\pounds(X) = \ln\sum_{T\epsilon X} Weight\ (T) = 1.6.$
Now, Contextanalysed metric

$$\Rightarrow \quad ¥(X) = c1 * I(Y) + c2 * R(Y) + c3 * B(Y)$$

$I(Y)_{virus}$= 4 and B $(Y)_{virus}$= 3 (besides while loop is running infinitely).Whereas, $I(Y)_{Regular}$= 3 and B $(Y)_{Regular}$= 2. [In both cases,R (Y) is zero].

⇨ **$¥(X)_{virus}$ = (1.0*4 + 1.0 * 0 + 0.1 *3) = 4.3**

⇨ **$¥(X)_{Regular}$= (1.0*3 + 1.0 * 0 + 0.1 *2) = 3.2**

So, the final profile metric **F_{virus}=μ_X= s1 * £(X) + s2* *¥(X),*
F_{virus}= (1.0*1.6 + 1.0*4.3)= 5.9
$F_{Regular}$ =(1.0*1.6 + 1.0*3.2) = 4.8

It can be easily shown that the complexity F_{virus} is greater than the $F_{Regular}$ for the above codes (F_{virus}>$F_{Regular}$).
Now the nature of the codes is also dependent on the CPU usage and memory usage parameters.

CPU Usage:
Here to calculate the usage of CPU in percentage, the below code is treated as virus code 2 and a simple code (calculate prime numbers) is considered as a regular code 2 for the experiment analysis (due to the lack of availability of virus code in public).

Virus code2:

```
for(i=1;i<1000;i++)
{
printf("Hello");
for(j=1;j<100;j++){
n=j/i;
printf("The value of n:%d",n);
}
}
```

To calculate the CPU time and CPU utilization it needs,

```
runTime = (end - start) / (double) CLOCKS_PER_SEC;
cpuutiliz=(runTime/(runTime+1.1))*100;
```

Where, 1.1 second is assumed as a context switching time of OS.

TABLE VII
CPU Utilization Analysis

Programs/Codes	CPU Time (Sec)	CPU Utilization (percentage)	File Size (Bytes)
Virus Code 1	Infinite	>> 80%	615
Regular Code 1	1.359	55.27%	618
Virus Code 2	5.39	83.05%	854
Regular Code 2	2.781	71.65%	854

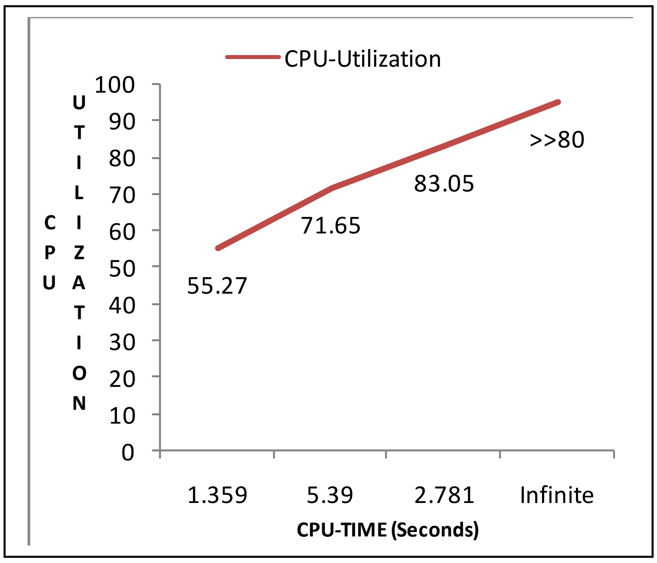


Fig.5. CPU-Utilization vs. CPU-Time

V. COMPARISON ANALYSIS

The proposed AMDS approach is a simple, flexible and probabilistic approach which is mainly applicable in the IaaS layer of cloud architecture. Several traditional intrusion detection systems based on ANN, genetic algorithm, data mining etc, are facing some challenges. This approach has some important advantages over these traditional malware detection approaches. The comparison analysis is listed in the Table VIII,

Table VIII. Comparison of AMDS with some traditional methods

Traditional Malware Detection Techniques	Challenges	Our Approach (AMDS)
Security attacks and solutions in Clouds [17]	(Due to the usage of FAT table, hypervisor method and virtual OS type) -Process time for the cloud provider is very high.	- Less processing time -Less expensive and low installation charge -Low Processing overhead -Very difficult for an attacker to intrude due to its integration with hardware based IaaS level - It is totally transparent to the user.
Malware Analysis in Cloud Computing: Network and System Characteristics [18]	-This approach is specifically focused on Kelihos attack. - Process time is high -Implementation is expensive	
Retrospective detection of malware attacks by cloud computing [20]	- These methods are only effective on Hadoop platform. - Process time is high due to a large number of files.	
Malware Detection in Cloud Computing Infrastructures [19]	- New and difficult to understand. - Required high time to extract unique signatures.	

VI. CONCLUSION & FUTURE WORK

In this paper several contributions on secure cloud via virtualization have been provided. It has proposed an effective and efficient advanced cloud security method that can detect the different malware attacks during VM to VM communication. It is also partially implemented on a popular architecture - Eucalyptus. This paper discusses the basic techniques in brief needed for the development of AMDS system.This technique is actually cheap, requires less memory space and provides less overhead compare to other well-known malware detection systems. It is totally transparent to the user. In future work, some priority based threats storage pool can be used to listing all the threats according to their harmful nature. It is only used to detect malicious codes. In future one prevention technique will also be proposed and implemented with this approach. This approach will be tried to fully implement in the Eucalyptus and some other well-known architectures.

ACKNOWLEDGMENT

Special thanks to Dr. Prof. N.K. Nagwani from the National Institute of Technology Raipur, whose comments and partial review improved the presentation of this article.

REFERENCES

[1] J. H. Cross , "GRASP Version 5.0 Graphical Representations of Algorithms, Structures, and Processes", Technical Report 96-15, September 29, 1996, Available: http://www.eng.auburn.edu/grasp.

[2] M. Christodorescu, R. Sailer, D. Lee Schales, D. Sgandurra, D. Zamboni, "Cloud Security Is Not (Just) Virtualization Security", CCSW'09, November 13, 2009, Chicago, Illinois, USA.ACM 978-1-60558-784-4/09/11.

[3] F. Sabahi, "Secure Virtualization for Cloud Environment Using Hypervisor-based Technology", International Journal of Machine Learning and Computing, Vol. 2, No. 1, February 2012, pp. 39-45.

[4] Lombardi F, Di Pietro R. , "Secure virtualization for cloud computing", Journal of Network and Computer Application(2010), doi:10.1016/j.jnca.2010.06.008.

[5] M.A.Vouk, "Cloud Computing – Issues, Research and Implementations", Journal of Computing and Information Technology – CIT 16, 2008, 4, doi:10.2498/cit.1001391, pp. 235-246.

[6] Q. Zhang · L. Cheng · R. Boutaba, "Cloud computing: state-of-the-artand research challenges", J Internet ServAppl (2010), Springer DOI 10.1007/s13174-010-0007-6, pp. 7-18.

[7] C. Williams, "Applications of Genetic Algorithms to Malware Detection and Creation", December 16, 2009, pp.1-13, Available: http://www.cs.colostate.edu/.

[8] Rashmi ,Dr.G.Sahoo, Dr.S.Mehfuz, "Securing Software as a Service Model of Cloud Computing: Issues and solutions", International Journal on Cloud Computing: Services and Architecture (IJCCSA) Vol.3, No.4, August 2013, DOI : 10.5121/ijccsa.2013.3401.

[9] D. C. Marinescu, "Cloud Computing: Theory and Practice", USA, December 6, 2012.

[10] P. Singhal, N.Raul, "Malware Detection Module using Machine Learning Algorithms to Assist in Centralized Security in Enterprise Networks", International Journal of Network Security & Its Applications (IJNSA), Vol.4, No.1, January 2012.

[11] Sarathy V., Narayan P., Mikkilineni Rao, "Next Generation Cloud Computing Architecture", Enabling Technologies: Infrastructure for Collaboration Enterprise (WETICE), 19th IEEE International Workshop, 28-30 June, 2010, pp. 48-53.

[12] D Swathigavaishnave and R Sarala, "Detection of Malicious Code-Injection Attack Using Two Phase Analysis Technique", International Journal of Computer Application, Volume 45- Number18, May 2012, pp.8-14.

[13] E. Mathisen, "Security Challenges and Solutions in Cloud Computing", 5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011), June 2011, pp. 208-212.

[14] M. A. AlZain, E. Pardede, B. Soh, J. A. Thom, "Cloud Computing Security: From Single to Multi-Clouds",45th Hawaii International Conference on System Sciences (IEEE), 2012, pp.5490-5499.

[15] Kundu, A. Banerjee, C. ; Guha, S.K. ; Mitra, A. ; Pal,C.; Roy, R., "Memory utilization in cloud computing using transparency", Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference (IEEE), 2010, PP. 22-27.

[16] "www.Gohacking.com", "www.vxheavens.com/lib/vbw06.html"& "www.viralforyou.blogspot.in/2011/07" - Virus codes.

[17] S.Singh, B.K.Pandey, R.Srivastava, N.Rawat, P.Rawat and Awantika, "Cloud Computing Attacks: A Discussion with Solutions", Open journal of Mobile Computing and Cloud Computing, Vol.1, 2014.

[18] A.K. Marnerides et al., "Malware Analysis in Cloud Computing: Network and System Characteristics", IEEE Globecom Workshops (GC Wkshps), 2013, pp.482-487.

[19] M.R. Watson, N.Shirazi, A.K. Marnerides, A.Maute and D. Hutchison, "Malware Detection in Cloud Computing Infrastructures",IEEE Transactions on Dependable and Secure Computing, 2015.

[20] H.M.Kanaker, M.M.Saudi and M.F.Marhusin, "A Systematic Analysis on Worm Detection in Cloud Based Systems", ARPN Journal of Engineering and Applied Sciences, Vol.10, No.2, 2015.