International Journal of Cloud Computing and Services Science (IJ-CLOSER)
Vol.1, No.3, August 2012, pp. 135~142
ISSN: 2089-3337

Regenerating Cloud Attack Scenarios using LVM2 based System Snapshots for Forensic Analysis

G. Geethakumari, Abha Belorkar
Department of Computer Science, BITS Pilani - Hyderabad Campus

Article Info

Article history:

Received July 10th, 2012
Accepted July 28th, 2012

Keyword:

Cloud Forensics
LVM2
Fuzzy Clustering
System Snapshots
Event Regeneration

ABSTRACT

Even though Cloud Computing has proved its utility and efficacy in many areas, security threats are a major obstacle in its widespread application. Cloud Forensics, with its existing equipment, has played an important role in improving our understanding of these threats, thereby contributing to the development of better and more robust cloud computing systems.In our earlier work, we introduced the use of fuzzy clustering techniques to detect and record malicious activities in cloudfor building strong and reliable evidences of the attacks. We now discuss the method in detail with certain essential aspects of its implementation. We also suggest ways to improve the time-complexity of the relevant back-end calculations.

Corresponding Author:

Abha Belorkar,
Computer Science,
BITS Pilani, Hyderabad Campus,
Jawahar Nagar, R. R. District, Hyderabad -500078.
Email: belorkar.abha@gmail.com

1. INTRODUCTION

The National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction" [15]. Thanks to cloud computing, we can now get computing resources required at all levels on a pay-as-you-use basis from the cloud service providers (CSPs) who also take care of the corresponding maintenance overhead [8]. Lack of computing resources is no longer considered a hindrance in experimentation. While these developments claim to have made life simple for corporate firms, security is still perceived as a major concern in cloud computing.

Of the several approaches taken to combat the potential security threats, Cloud Forensics is used to analyze and investigate the nature of cloud attacks in an attempt to recover from their damage. Several tools such as log files and system snapshots are used to extract information about the attacks. However, these methods only lead to a blurred view of the events [2]. Also, since the network timing protocol is not applied to cloud computing, we have inevitable differences in the time stamps at the client's and CSP's end which make it more difficult to correlate various instances at both ends. Hence, traditional techniques fail to give a clear picture of the exact course of attack or the sequence of manipulation of data and processes involved. This work aims at providing a concrete and sequenced evidence of all events that pose threats to the security and privacy of the data and computations entrusted with the cloud.

To achieve this objective, we need to take a series of file-system snapshots (periodically) that are capable of storing the exact sequence of manipulation in system states. Consequently, we need to address the following important issues:

- The snapshot should contain the detailed information of all components involved or affected in the attack and should be globally consistent.
- The system downtime while taking these snapshots should be minimal so as to avoid freezing of communication and computations in the VM.
- We should ensure that *only* the attack and *whole* of the attack is recorded.

To address the first two issues, we had proposed, in our previous work [1], the use of VNsnap which is an excellent tool that takes system snapshots of an entire VNE from outside the virtual machines (VMs) at layer 2 switches. VNsnap(a system built in [3]),using Mattern's distributed snapshot algorithm based on message coloring [13], provides all the information about concerned components through globally consistent snapshots. However, due to certain implementational problems, we went ahead with the rsnapshot package in LVM2 instead. It provides an excellent alternative mechanism to take periodic snapshots of VNE in Linux-based operating systems.Using this setup to take periodic snapshots, we propose a fuzzy-clustering based algorithm for recording the cloud attacks accurately (*only* the attack and *whole* of the attack).

1.1. LVM2:

LVM (Logical Volume Manager) is a free package which provides the facility of taking snapshots with no downtime. This allows the administrator to create a new block device which presents an exact copy of a logical volume, frozen at some point in time. In LVM2 snapshots are read/write by default (as opposed to read-only snapshots provided by LVM1). The advantage is that we can snapshot a volume, mount the snapshot, and try experimental programs that change files on that volume. The underlying device mapper operations take place transparently, without applications or file-systems being aware that their underlying storage is moving[17]. A package rsnapshot which assumes LVM support was used for taking the snapshots.

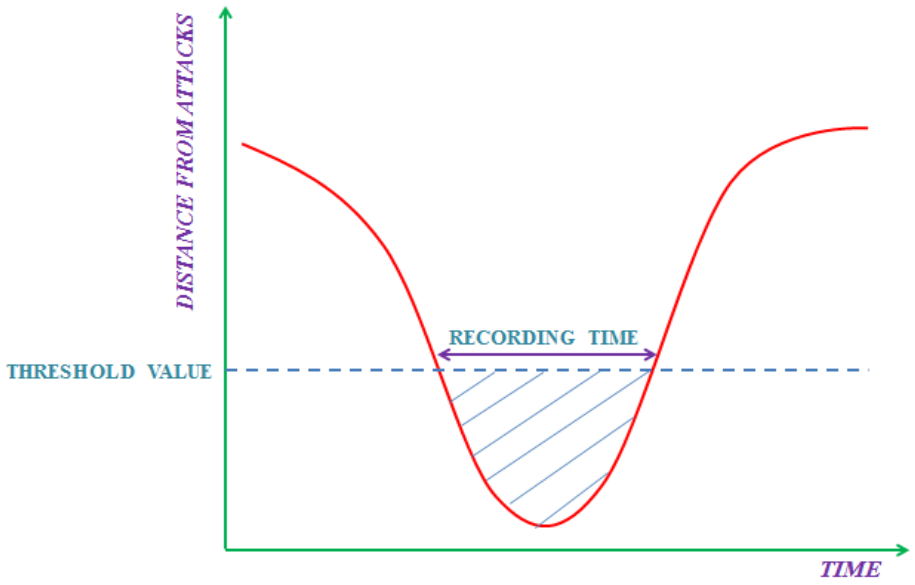

Figure 1. Snapshot Recording Duration

1.2. Recording the attacks correctly:

In our work, we assume that enough documentation is available about the cloud attacks that have previously taken place in various similar environments. Generally, these documented attacks are classified using hard clustering techniques such as partition clustering (organized category wise as disjoint sets of attacks) and hierarchical clustering (organized into further subdivisions inside categorized sets) [16]. This means that each new attack is placed in an existing 'partition' or 'subdivision' depending on some fixed or predetermined characteristics. However, partition clustering is usually too simplistic to qualify as an effective classification technique, while hierarchical clustering introduces painful dependencies among the attack sets. Moreover, after studying the complexity of cloud attacks, we found the need to consider the degrees of

similarity between the new attack and all existing sets rather than complete assignments to one of the fixed sets or clusters of attacks. Thus, we would like to rely on algorithms which would determine the degree of belongingness of a new entity to each of the existing sets rather than those which would assign it to only one of the predetermined sets [1]. Since fuzzy clustering is a soft clustering technique which associates a set of membership levels with each new entity [6], we introduce their application to the classification of these attacks such that their effectiveness as references or tools in cloud forensic analysis is increased. By doing this, we actually take into consideration the overall impact of all similar attacks, thus ensuring better accuracy in the results. We employ a continuously running background calculation module in the VNE, which detects the similarity between the current code modules in execution in each of the VMs and the previously documented attacks. However, what we measure is the dissimilarity in the form of a parameter called 'distance' (defined in the next section). We experimentally determine a threshold value below which this dissimilarity can be considered malicious. The recording of system snapshots continues only as long as the quantified dissimilarity [6] remains fallen below the threshold value; thus ensuring that the entire attack is recorded (Figure 1).

Extending our previous work [1], we explain the implementational details and also propose a computationally efficient algorithm to execute the back-end calculations involved.

2. RESEARCH METHOD

The detailed methodology for taking the desired series of system snapshots can be given as:

2.1. Defining attacks as clusters:

The available documentation on previous attacks should first be organized into small sets of primitive operations such that each set qualifies as an individual attack. We treat each of these sets as a single-point (or single-object) cluster. Each point or attack is then represented as a vector in n-dimensional space where each dimension represents a quantifiable characteristic of the code such as memory usage, processing power requirement, bandwidth usage, order of complexity, etc. Thus each attack X_p can be assumed to have spatial co-ordinates $(X_{p1}, X_{p2}... X_{pn})$.

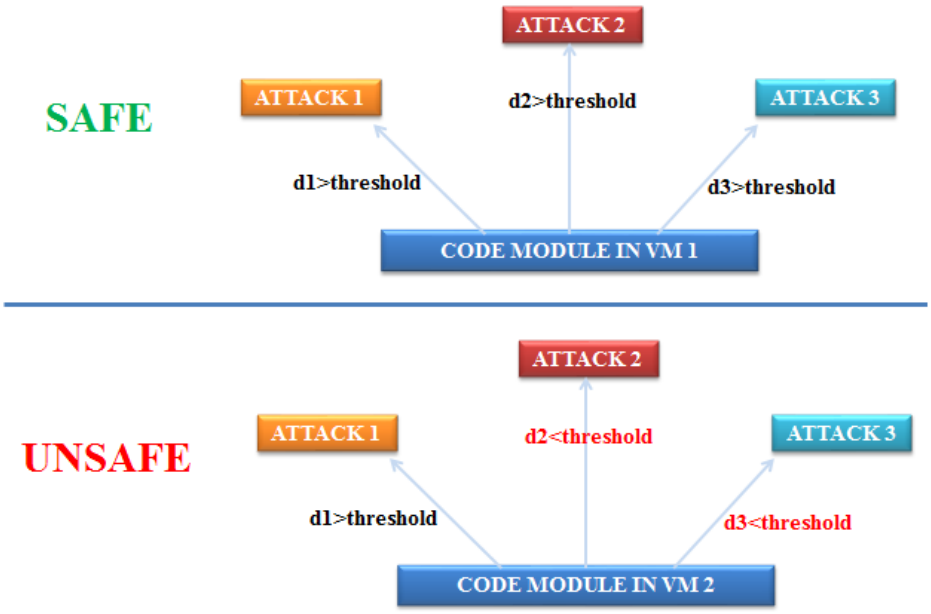

Figure 2. Comparison of current code module with documented attacks

2.2. Assigning weights to dimensions:

It is also important that we take into consideration the relevance of each dimension defined in the space. Therefore, we assign a weight w_i to the i^{th} dimension to quantify its importance in determining the similarity (or dissimilarity) between two codes. This weight should be determined experimentally. For example, if we find that similarity in order of complexity is twice as important as similarity in memory usage to infer that two codes are similar, then the weight assigned to order of complexity would be twice of that assigned to memory usage.

2.3. The current code module distance:

In every VM, the code module in execution is first identified. Each of these modules is again assigned co-ordinates in the same n-dimensional space based on its own characteristics. However, in this case, these code modules are a function of time and so are their co-ordinates. A code module $Y_q(t)$ in the q^{th}VM, for example, will have co-ordinates: $(y_{q1}(t), y_{q2}(t), \ldots, y_{qn}(t))$. We then calculate the "distance" of the code module from each of the clusters (attacks). The distance between an attack Xp and a code module $Y_q(t)$ which is given as:

$$D_{pqi}(t) = \sqrt{(\Sigma(w_i(x_{pi} - y_{qi}(t)))^2)} \quad \ldots i \text{ varies from } 1 \text{ to } n. \quad (1)$$

Note that this distance signifies *dissimilarity* [5] between the p^{th} attack and the code module of the q^{th} VM at time *t*. By assigning weights to the dimensions, we have ensured that the inferences about the measured degree of similarity or *dissimilarity* are correct

2.4. Determining the threshold value:

We need to determine a threshold value for distance between attacks and code modules below which the system may be harmed (Figure 2). For this, we experimentally find out the minimum distance that should be maintained from any attack in order for the system to be safe [1]. This experimentation involves:

1. The simulation of commonly occurring attacks (as available in the documentation)
2. Observing the quantified values of all their characteristics (in the form of their co-ordinates) such as memory usage, bandwidth usage, processing power requirement, etc. (as mentioned in Section 2.3)
3. Simulation of numerous harmless code modules on the VMs and comparing their co-ordinates with thoseobtained in Step 2.
4. Finding the minimum of differences between the code modules and attacks in each dimension. Say, the minimum of differences between the harmless code modules and the attacks with respect to dimension *'i'* is m_i, then we obtain the following values for all n dimensions: $m_1, m_2, \ldots m_n$.
5. Calculating the threshold:
   Threshold $= \sqrt{\sum(w_i m_i)^2}$ $\ldots i$ varies from *1* to *n*. (2)

Following are the characteristics of the threshold value obtained by the above procedure:

1. The threshold value is high enough to ensure that accidental similarities with attacks are generally not accounted for.
2. The value is low enough to ensure maximum probability of the malicious code modules being tracked.

2.5. The snapshot duration:

As mentioned before, we employ a continuously running calculator for measuring the time-dependent distances between code modules in all VMs and the attack sets (as in equation (1)). We start taking periodic snapshots programmatically in a time loop using rsnapshot when the distance of any of the code modules from some attack decreases below the threshold value and stop the process when all the distances are found to be above the threshold. Thus the entire attack is recorded (Figure 1).

2.6. Regenerating the event:

The attack can be regenerated by restoring the system to each of the snapshots (programmatically again) in the same sequence as that of their creation. However, an isolated snapshot of the system should be taken before this regeneration process so that the system can be restored back to its original position once the attack is replayed.Theentire process is effectively represented in Figure 4.

.

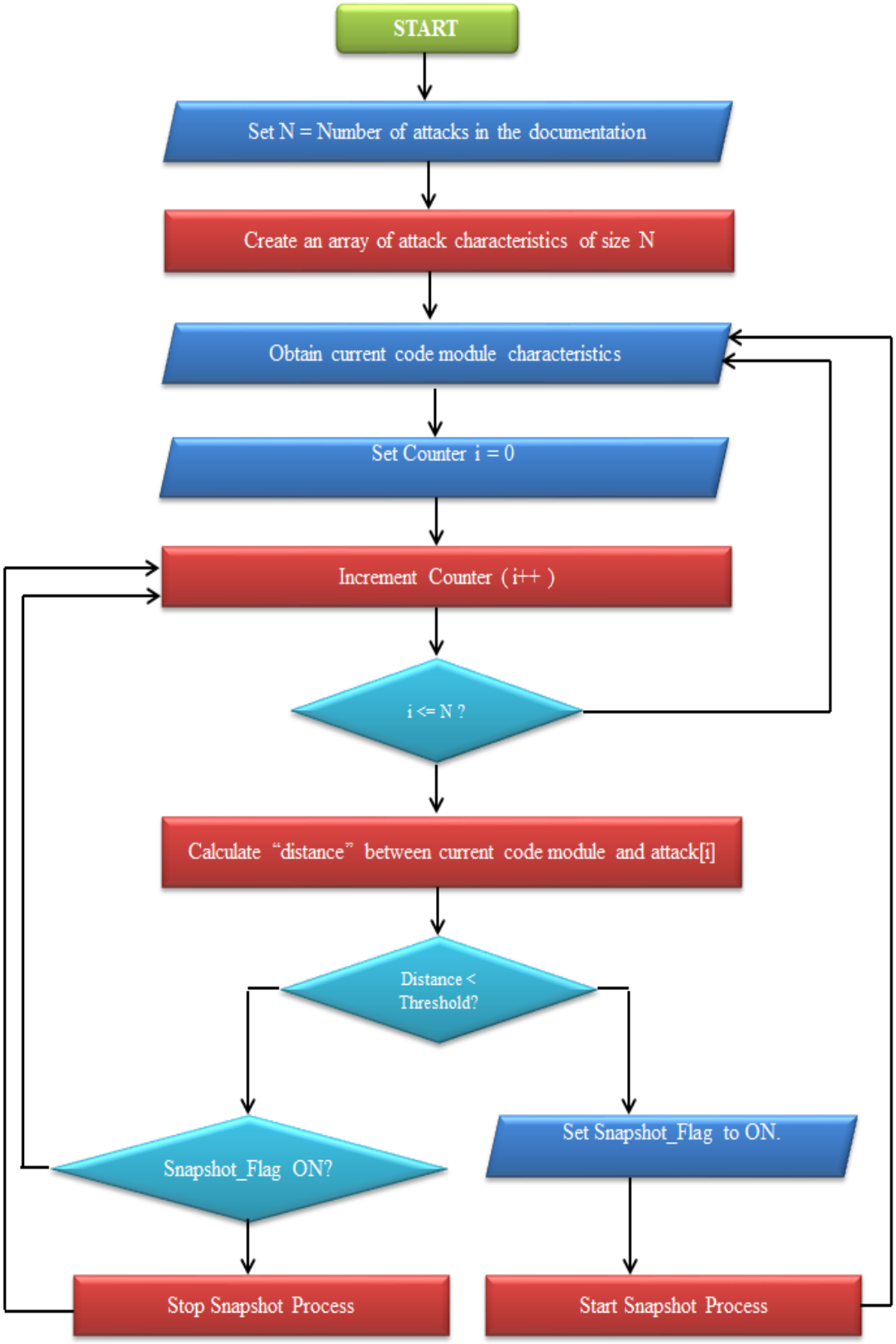

Figure 4. The Algorithm illustrated

3. IMPLEMENTATION

The implementation of the idea can be described as below:

3.1. Defining attacks as clusters:

We simulate a few standard cloud attacks and note down their requirement in terms of memory (dimension 1) and processing power (dimension 2) (more dimensions could be added in a more advanced implementation). Thus, each attack $X_p$is assumed to have spatial co-ordinates (X_{p1}, X_{p2}).

3.2. Assigning weights to dimensions:

To take into consideration the relevance of each dimension defined in the space, we assign a weight w_i to the i^{th} dimension to quantify its importance in determining the similarity (or dissimilarity) between two codes (points in space). To start with, we assume that processing power is twice as important as memory usage in order to ascertain similarity with the attack.We then define two variables (number of variables = number of dimensions considered) namely Weight_X1, Weight_X2 which will store the corresponding weights (in this case, *Weight_X1* = 1, *Weight_X2* = 2). After this random value allocation, the weights can be revised using computational models such as Artificial Neural Networks etc.

Now we regenerate the co-ordinates after multiplying the relevant quantities (memory and processing power in this example) with their respective weights: $X_{p1}=X_{p1}$*Weight_X1,$X_{p2}= X_{p2}$* Weight_X2. After this process, for *n* attacks, we have the points: (X_{11}, X_{12}), (X_{21}, X_{22}), … (X_{n1}, X_{n2}).

When this is done, find and store:

1. *X1_min* = Minimum$(X_{11}, X_{21, \dots} X_{(n-1)1}, X_{n1})$
2. *X1_max* = Maximum$(X_{11}, X_{21, \dots} X_{(n-1)1}, X_{n1})$
3. *X2_min* = Minimum$(X_{12}, X_{22, \dots} X_{(n-1)2}, X_{n2})$
4. *X2_max* = Maximum$(X_{12}, X_{22, \dots} X_{(n-1)2}, X_{n2})$

We now know the range of co-ordinates (the upper and lower bound for each dimension) in which the attacks are spread. In the case of two dimensions, this translates to a rectangle confining all the points. We have defined the least counts for both the dimensions *LC_1*, *LC_2* such that all attack co-ordinates will be truncated to appropriate accuracy. We took *LC_1* = *LC_2* = 0.1, such that all $X_{1i}$and X_{2i}co-ordinates will be of the form: *a.1, b.2…k.n.* After this truncation, we simply create a two-dimensional array whose dimension 1 unit is *LC_1* and dimension 2 unit is *LC_2*. Now, we just need to assign 1's in the array for the elements whose index matches with the attack-co-ordinates (that is co-ordinates of the previously documented attacks) and 0's in the remaining array elements. This implementation reduces the computational complexity by an order of *n* and this is explained in the next section.

3.3. The current code module distance:

On identifying the code module in execution in each VM, co-ordinates are assigned to them in the same 2-dimensional space based on their characteristics. However, because these code modules are a function of time, they have to be defined instantaneously. So at a given instant, suppose that the co-ordinates of the current operation are: (Y_{q1}, Y_{q2}). We then calculate the "distance" of this code module from each of the clusters (attacks). The distance between an attack X_p and a code module $Y_q(t)$ which is given as per equation (1) (in section 2.3). However, note that, to calculate this distance for n attacks would require *O(n)* time which would not be in accordance with our claim that the background calculation is light-weight. So, instead, we simply mark the rectangle traced by the following points:

1. $(Y_{q1}$ - *threshold*, Y_{q2} - *threshold)*
2. $(Y_{q1}$ - *threshold*, Y_{q2} + *threshold)*
3. $(Y_{q1}$ + *threshold*, Y_{q2} - *threshold)*
4. $(Y_{q1}$ + *threshold*, Y_{q2} + *threshold)*

Then check only those array elements that are enclosed within this rectangle (the concept of threshold is discussed in section 2.4). Since the area of the rectangle is always fixed and independent of *n*, we need to make a constant number of comparisons and check if any array element is 1. If that is the case, we can report an unsafe state. If all elements are 0s, we can ensure that the operations are free from malice. Thus, we have reduced the time complexity of the background calculation from *O(n)* to *O(1)* at the cost of increased space complexity which presently is not a computational efficiency issue.

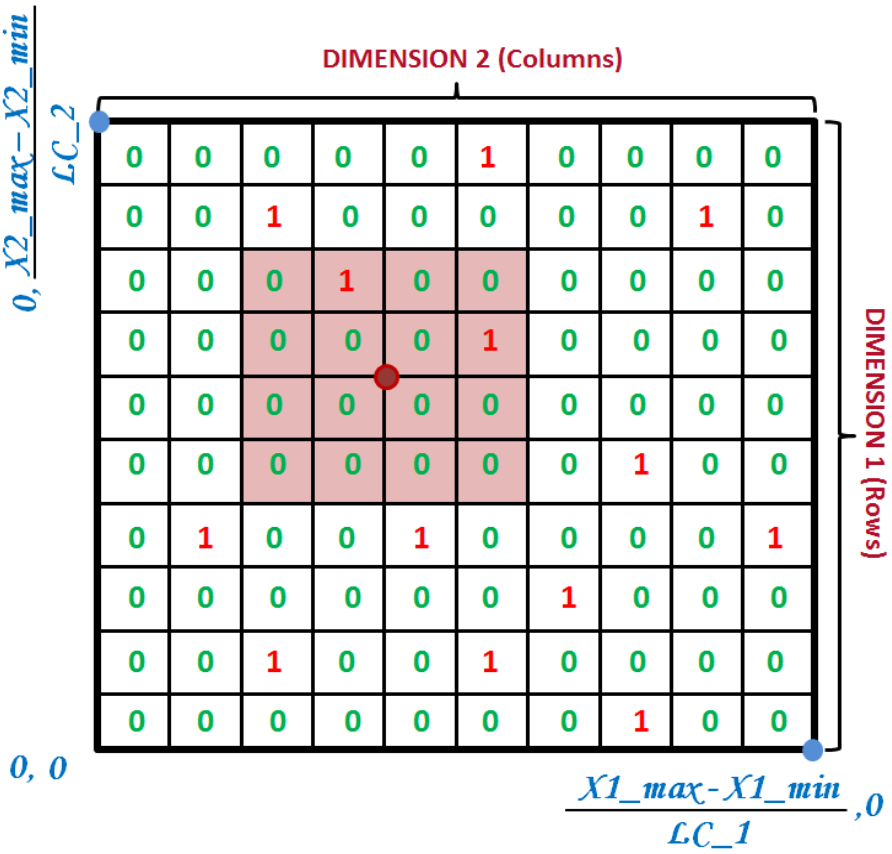

Figure 3. Two dimensional array for storing co-ordinates of attacks

3.4. Determining the threshold value:

As mentioned earlier, we need to determine a threshold value (determined as in section 2.4) for *distance* between attacks and code modules below which the system may be harmed (Figure 2).

3.5. Taking the snapshots:

We use LVM2 with Xen to take the snapshots of the affected file system(s). The snapshot recording will start as soon as an unsafe state is reported and stopped when the system reaches a safe state (section 2.3).

4. CONCLUSION

As mentioned in our earlier work, the resulting evidence is sequenced, integrated and much stronger. It provides clearer insight into the strategy of the attacks and can even be used for qualitative evaluation of the IDS/IPS systems. Morever, the method described for performing the back-end calculations (section 3.3) with lesser time complexity and greater efficiency contributes furthur to the feasibility of its application.

The potential of this work to be applied to more practical and popular cloud application environment relies heavily on the tools employed for accurate and exhaustive identification of all the parameters (variables) that characterize malicious activities in the cloud. Also, the corresponding weights of these parameters may be assessed more accurately if advanced Machine Learning techniques are used. Similar appropriate techniques may also be used for a more precise detection of the *threshold* value.

REFERENCES

[1] Abha Belorkar, G. Geethakumari, “Regeneration of events using system snapshots for cloud forensic analysis”, IEEE Xplore (INDICON, IEEE India Conference 2011).
[2] T. V. Lillard, C. P. Garrison, C. A. Schiller, J. Steele, and J. Murray, “Digital Forensics for Network, Internet and Cloud Computing”, Elsevier, 2010.
[3] A. Kangarlou, P. Eugster, D. Xu, “VNsnap: Taking Snapshots of Virtual Networked Environments with Minimal Downtime,” IEEE, 2009.

[4] J. O. Nehinbe, "A framework for evaluating clustering algorithms," 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010.
[5] J.O. Nehinbe "Guessing strategy for improving Intrusion Detections", Computer Science & Electronic Conference, proceedings of IEEE, London, UK, 2010.
[6] S. Miyamoto, H. Ichihashi, and K. Honda, "Algorithms for Fuzzy Clustering," Springer-Verlag Berlin Heidelberg, 2008.
[7] S. Zimmerman and D. Glavach, "Cyber forensics in the cloud," in IANewsletter, vol.14, number 1, Winter 2011.
[8] A. T. Velte, T. J. Velte, and R. Elsenpeter, "Cloud computing – A practical approach", TMH, 2010.
[9] B. Grobauer and T. Schreck, "Towards incident handling in the cloud: Challenges and Approaches," Proceedings of the 2010 ACM workshop on Cloud computing security workshop, New York, USA, 2010.
[10] X. Fu, Z. Ling, W. Yu, and J. Luo, "Cyber Crime Scene Investigations (C2SI) through Cloud Computing," IEEE 30th International Conference on Distributed Computing Systems Workshops, 2010.
[11] A. Kangarlou, D. Xu, et al., "In-Network Live Snapshot Service for Recovering Virtual Infrastructures," IEEE Network, July/August 2011.
[12] L. Lamport, and K. ManiChandy, "Distributed snapshots: Determining global states of distributed systems," ACM Transaction on Computer System, 1985.
[13] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," in Journal of Parallel and Distributed Computing, 18:423–434, 1993.
[14] E. Stewart, J. Kennedy, "The sustainability potential of cloud computing: Smarter Design", in Environmental Leader: Environment and Energy Management News, July 20, 2009.
[15]P. Mell, T.Grance, "The NIST Definition of Cloud Computing", Version 15, July 10, 2009.
[16]Y. Zhao, C. Chi, C. Ding, "Analysis of data clustering support for service", IEEE 2nd international conference on Software Engineering and Service Science (ICSESS), 2011.
[17] http://www.softpanorama.org/Internals/Unix_filesystems/snapshots.shtml

BIBLIOGRAPHY OF AUTHORS

Dr. G. Geethakumari is presently Asst. Professor, Dept. of Computer Science and Information Systems at BITS-Pilani, Hyderabad Campus. Prior to joining BITS, she was a faculty member at CSE Dept, NIT-Warangal. Her research interests include: grid computing, information security, access control modelling, mobile application security, parallel computing, cloud computing, and cloud security. She has many international publications to her credit. She has been a Program Committee member for many international conferences and is also in the book review panel of publishers such as TMH and Springer.
Dr. Geetha has been in the forefront of technical activities at BITS-Pilani, Hyderabad Campus. She has held various positions such as Faculty Advisor, Computer Science Association in BITS-Pilani, Hyderabad Campus. She is a Member, IEEE, and IEEE Computer Society Member as well as Member, ACM.

Ms. Abha Belorkar is currently an undergraduate student of Computer Science at BITS Pilani, Hyderabad Campus. Her research interests include Network Security, Cryptography, and Algorithms.