Scalable Computing: Practice and Experience
Volume 11, Number 4, pp. 373–383. http://www.scpe.org
ISSN 1895-1767

IMPLEMENTING SELF-PROTECTION IN DISTRIBUTED GRID ENVIRONMENT*

INDERPREET CHOPRA† AND MANINDER SINGH‡

Abstract. Grids encourage the dynamic addition of resources that are not likely to be benefited from the security that static, centrally administered commercial firewall's or IDS's provide. On the contrary, Grids need to enforce their own security policies that are self automated, as the manual management techniques are time consuming, insecure and error prone. This paper presents a new self-protection model—SPM based upon few principles of genetic algorithm.

Key words: genetic algorithm, SPM, Snort, GT4

1. Introduction. The Grid is constantly growing and it is being used by more and more applications. However the grid services are increasing in complexity [4, 5] in modern era. These can vary over a large number of parameters: from the network (topology, speed and security policies), the resources (number of machines, processors and machine usage) to the scheduler (fork or more complex once) [3].

The complexity of today's distributed computing environment is such that the presence of bugs and security holes is unavoidable [2]. Before a user runs an application,he needs assurance that the machine has not been compromised which could subject his application from being stolen. When a user's job executes, the job may require confidential message passing services [9].Hence, security is a major concern for any IT infrastructure.

Overall,most of the problems stem from the fact that human administrators are unable to cope up with the amount of work required to secure the computing infrastructure properly in the age of Internet [12]. Autonomic systems and self-managed environments provide security policies establishing promising trust between the server and the client. Hence, autonomic computing, [7, 8] presented and advocated by IBM, suggests a desirable solution to this problem [6].

Self-Protection enables the system with an ability to secure itself against attacks i. e. detect illegal activities and trigger counter measures in order to stop them. It also helps to overpower the limitations of manual management of the system i. e. slow speed, increasing chances of errors and unmanageability by human administrators.

This paper proposes a Self-Protection Model (SPM)- an agent enabling autonomic computing providing a promising solution to the system management troubles caused by increased complexity of large scale distributed systems. SPM is based upon few implications of the genetic algorithm (GA). GAs are robust, inherently parallel, adaptable and suitable for dealing with the classification of rare classes. Moreover, due to its inherent parallelism, it offers a possibility to implement the system without requiring any additional resource. This new model adopts intelligent agents to dynamically organize system management with centralized control. At the system level each element contributes its capabilities on the functions of system management and cooperate with each other to implement autonomic computing for the grid system. Cooperative works are organized by dynamically associated relationships among autonomic elements (agents), including acquaintance, collaboration and notification.

This self-organized model is more suitable for the grid environment as it is characterized by distributed, open and dynamic properties.

2. Self-Protection. A self-protecting system helps to detect and identify hostile behavior and take autonomous actions to protect itself against intrusive behavior. The main goal of self-protection system is to defend grid environment against malicious intentional actions by scanning the suspicious activities and react accordingly without the users being awared that such protection is in process [2].

2.1. Current Security Vulnerabilities. Kind of vulnerabilities, against which we need self-protection are:

QoS Violation Grid service must provide the best service with respect to the service level agreement (SLA) signed between the user and the provider [15]. Different types of Qos violation attacks include:

*This work is done at Thapar University, India to automate the security process in grid computing.

†Research Scholar, Computer Science Department, Thapar University, Patiala, India (inderpreet@thapar.edu).
‡Associate Professor, Computer Science Department, Thapar university, Patiala, India (msingh@thapar.edu).

- *Dropping Packets:* In these types of vulnerabilities, attacker chooses to drop the packets flowing through the grid networks by taking control over some component in the grid network.
- *Delaying Packets:*In this type of attacks, attackers reduce the flow rate of the packets so as to reduce the effective QoS.

DoS attacks These continue to be the main threat. In such attacks, a set of attackers generates a huge traffic, saturating the victim's network, and causing significant damage. Sun Grid was brought to its knees by a distributed denial-of-service (DDOS) attack [19], necessitating an emergency login procedure change. While grid computing may very well revolutionize enterprise computing, the incident underscores the security risks that could prove quite harrowing for enterprises that rely on grid computing [17].These include:

- *TCP floods:* A stream of TCP packets with various flags set are sent to the victim IP address. The SYN, ACK, and RST flags are commonly used.
- *ICMP echo request/reply (e.g., ping floods):* A stream of ICMP packets are sent to a victim IP address.
- *UDP floods:* A stream of UDP packets are sent to the victim IP address.
- *Source IP address:* In some cases, a false source IP address, a method commonly called IP spoofing, is used to conceal the true source of a packet stream. In other cases, IP spoofing is used when packet streams are sent to one or more intermediate sites in order to cause responses to be sent toward a victim. The latter example is common for packet amplification attacks such as those based on IP directed broadcast packets (e.g., "smurf" or "fraggle").
- *Source/destination ports:* TCP and UDP based packet flooding attack tools sometimes alter source and/or destination port numbers to make reacting with packet filtering by service more difficult.
- *Other IP header values:*At the extreme, we have seen DoS attack tools that are designed to randomize most all IP header options for each packet in the stream, leaving just the destination IP address consistent between packets.

Insider Attacks Attacks from the inside carry the potential for significant damage that can reveal or even exceed the damage caused by external forces [13]. As an integral and trusted member of the organization, the perpetrator carries valid authorization and typically enjoys relatively unchallenged presence and movement within the organization's IT infrastructure. The attacks typically target specific information and exploit established entry points or obscure vulnerabilities. In many respects, insider attacks can be more difficult to detect than penetration attempts from the outside.

Remote to local(R2L) Attacks in which an unauthorized user is able to bypass normal authentication and execute commands on the target [9].

User to root (U2R) Attacks in which a user with login access is able to bypass normal authentication to gain the privileges of another user, usually root. This can also be the case when the user try to use more or some other resources assigned to him [9].

2.2. Related Work. Few of the self-protection tools/techniques available to handle the attacks on grid system are:

- **Firewall**: The firewall is configured to filter out the packets to enter inside the grid environment. Firewalls basically separate the protected network from the unprotected network [26] [27]. They screens and filters all connections coming from internet to the protected (corporate) network and vice versa, through a single concentrated security checkpoint.
- **Intrusion Detection System**: Intrusion Detection Systems add an early warning capability for the suspicious activity that mostly occurs before and during an attack. Intrusion detection systems (IDSs) often work as misuse detectors, where the packets in the monitored network are compared against a repository of signatures that define characteristics of an intrusion.
  Many grid based IDS systems have also been conceived, designed and implemented. [24] The basic components for grid based IDS systems include: Sensors, which are able to monitor the state of grid systems. The information collected by sensors are then collected and analyzed by IDS system like SNORT [25]. Based upon the analysis, alarm is raised.
  Many approaches for Grid based IDS prevailed in the market. Kenny and Coghlan [28] describe a system that allows the querying of log files through the Relational Grid Monitoring Architecture (R-GMA), which can be used to build a Grid-wide intrusion detection system. Their implementation of this system, called SANTA-G (Grid enabled System Area Networks Trace Analysis), queries Snort log files by using SQL. SANTA-G is composed of three elements: A Sensor, a QueryEngine and a Viewer

GUI. Schulter et al. [29] describes different types of IDS systems. The authors point out that present Grid-IDS approaches do not fulfil the important qualities (completeness—recognition of all attack-types, scalability and Grid-compatibility) for protection of Grid systems. They propose a high-level GIDS that utilizes functionality of lower-level HIDS and NIDS provided through standard inter-IDS communication.
Michal Witold [30] in his thesis investigates the possibility of Grid-focused IDS. The main stress is put on feature selection and performance of the system. Leu and Li [31] proposed Fault-tolerant Grid Intrusion Detection System (FGIDS) which exploits grid's dynamic and abundant computing resources to detect malicious behaviors from a massive amount of network packets. In FGIDS, a detector can dynamically leave or join FGIDS anytime.

3. Self-Protection Model (SPM) for Grid. We have discussed the various problems and the related work that has already been done to avoid the grid system services to be compromised. All of these areas are under constant investigation by researchers and have been so for a long time. There is still no grid middleware that is intelligent enough to handle the failures and faults automatically. We are trying an attempt to unify related research on these by providing the self-management environment to manage the security and faults in grid. SPM model has been given in this paper, providing a way for automatically creating a security wall to protect against external attacks like DDoS. Main advantages of using SPM are:

1. Robust Agent based architecture is used in SPM.
2. Genetic approach used by filter agent to automatically monitor the malicious activities. Many features of GA make it very suitable for intrusion detection. Like parallelism, self-learning capabilities, and the fact that initial rules can be built randomly so there is no need of knowing the exact way of attack machinery at the beginning.
3. Easy to use user interface for job submission and monitoring.

3.1. SPM Architecture. SPM has the layered architecture based upon MVC. MVC stands for Model (Database layer), View (User Interface) and Controller (Business Logic). In the (Figure: 3.1), the application layer is View, the agents and autonomic unit is Controller and the database layer defines the model. Advantages of using this architecture is to achieve:

- Flexible: Can be used with any database, application server with very small effort to change.
- Customizable: Add new features and screens without affecting any existing functionality.

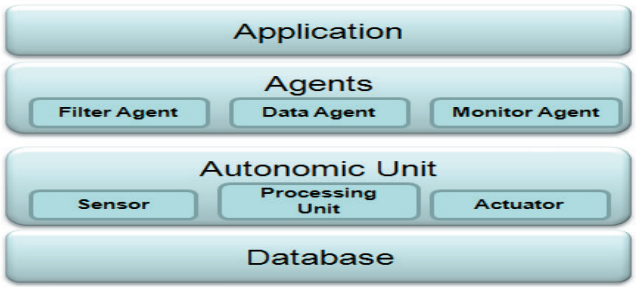

FIG. 3.1. *SPM Architecture*

In this section the details of SPM architecture are described.
Application Layer: This layer provides the user interface for the user to interact with the grid environment. This layer authenticates the originality of the user before letting the user to access the grid. After the user is authenticated, it shows the different options to the user to submit, monitor and cancel the job.
Agents Layer: Agent layers consists of three types of agents and each perform their respective tasks. These are:

- *Filter Agent*: It helps to safeguard the system from unwanted requests and various security threats (Figure: 3.2). Only the authenticated users can submit the jobs. These agents are configured to allow only a certain type of traffic to pass through them. Whenever they detect the illegal packets, they simply discard them. After all inspection is done, they publish the job request to the queues. For this agent, we use Snort to setup the distributed IDS [33][34]. To make filter agent intelligent enough to handle new threats, we have used the Genetic Algorithm (GA) approach to define new rules.

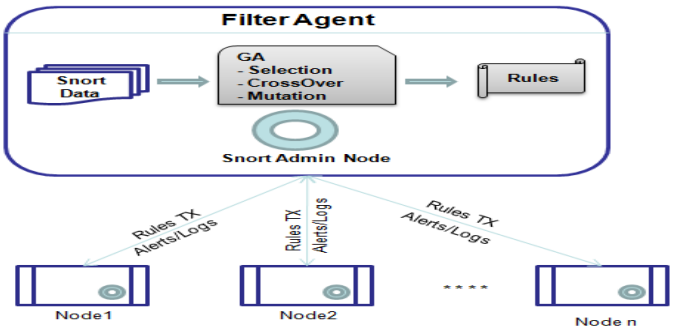

FIG. 3.2. *SPM Filter Agent*

Genetic Algorithm

GA is based upon the principles of evolution and natural selection (Figure: 3.4). GA converts the problem in specific domain into model by using chromosome like data structure and evolve the chromosome using selection, crossing over (Figure: 3.3) and mutation operators [10]. The process of genetic algorithm usually begins with the population of chromosomes representing the gene pool. Different positions of each chromosomes are encoded as bits, characters or numbers and referred to as *Genes*. Genes can be randomly changed within a range during evolution of the species. An evaluation function is used to calculate the "goodness" of each chromosome. During evaluation, two basic operators—Crossing Over and Mutation are used, simulating the natural reproduction and mutation of species. In our

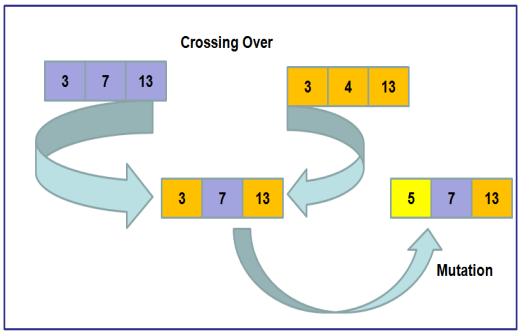

FIG. 3.3. *Crossing Over and Mutation*

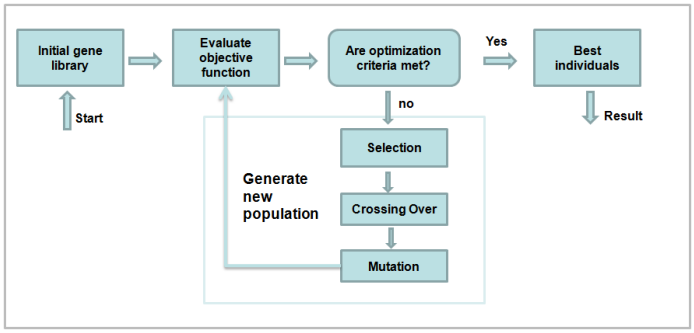

FIG. 3.4. *Structure of GA [11]*

model we use this approach for intelligently defining the new rules which are later transferred to various child resources attached to the central node (Figure: 3.5). Our filter agent GA for snort work like this:

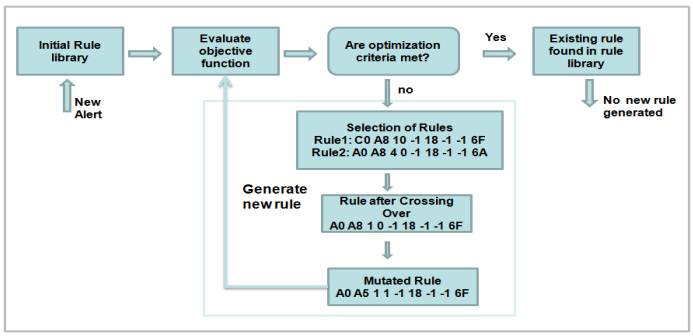

FIG. 3.5. *GA based Rule Generation*

```
SPM Filter Agent GA()
{
      Initialization;
      Evaluation;
      while termination criterion has not been reached
      {
            Selection and Reproduction;
            Crossover;
            Mutation;
            Evaluation;
      }
}
```

Initialization: It involves setting the parameters for the algorithm and creating the first generation of chromosomes. In this benchmark, each chromosome is represented by the set of genes. And the genes contains the description of the rules Let's take an example. Consider one sample rule *alert tcp any any -> 192.168.1.0/24 111* and it is represented as hexadecimal chromosome set (C0 A8 10 -1 18 -1 -1 6F)
Evaluation: Each of the chromosomes in a generation is evaluated for the selection process. This is accomplished by looking up the success rate of the rule representing the chromosome. If the rule is able to find the anomalous behaviour, its fitness is increased else decreased.
Selection and Reproduction: We use the simplest of all i. e. roulette wheel selection[32]. In roulette wheel selection, individuals are given a probability of being selected that is directly proportionate to their fitness. Two individuals are then chosen randomly based on these probabilities and their capability of producing offspring.
Crossover: In the crossover phase, all of the chromosomes with similar lengths are paired up, and with a probability they are crossed over. The crossover is accomplished by randomly choosing a site along the length of the chromosome, and exchanging the genes of the two chromosomes for each gene past this crossover site.Say if we have the two parent chromosomes represented by (C0 A8 1 0 -1 18 -1 -1 6F) and (A0 A8 4 0 -1 18 -1 -1 6A), then the child chromosome can be (A0 A8 1 0 -1 18 -1 -1 6F)
Mutation: After the crossover, for each of the genes of the chromosomes, the gene can be mutated. With the crossover and mutations completed, the chromosomes are once again evaluated for another round of selection and reproduction.We have skipped the details related to performance and working of SPM Filter agent GA to keep things simple for SPM understanding.

- *Data Agent*: Picks the job request from the queues. Store the data into the database. Jobs can be from high priority clients or normal priority. This information is also saved into the jobs database so as to later use this information for deciding the execution based upon the priority. Also these agents are responsible for data transfer from the user machine to the centralized server, so that when the execution unit needs the data for executing submitted jobs, it can pick it from the one single location. For data transfer, ftp protocol is configured that requests the users to upload the necessary data that their job will require for execution. The packets while transfer has to pass through the filter agent that keeps on scanning each packet for any harmful content.
- *Monitor Agent*: It keeps on monitoring the participating nodes in the grid. With the information thus obtained, it updates the resources states in the database. This monitor agent keeps on running and update information about the resources after certain interval of time. If a system is using some

metascheduler, then this agent also have the feature to pick the information from that scheduler rather than from the individual head nodes.

Autonomic Unit Layer: This layer will help in providing the autonomic capabilities to self heal and self-protect the system without any user intervention. This layer consists of:

- *Sensor*: Sensor keeps on monitoring any changes in the jobs or resources database. Whenever any change is introduced, it updates the knowledge bank of the processing unit.
- *Processing Unit*: PU deals with the analysis and planning of job execution. It takes into consideration the job requirements and the available resources wrt those requirements. It is the PU, which based upon the users priority level, makes the plan for job execution. Another things taken into consideration while making the plan are: job status, job resource requirement, time of submission, state of resources involved, etc
- *Actuator*: This takes the plan from the PU, and started working to execute it. Another task that the actuator performs in the updation of the job state in the database.

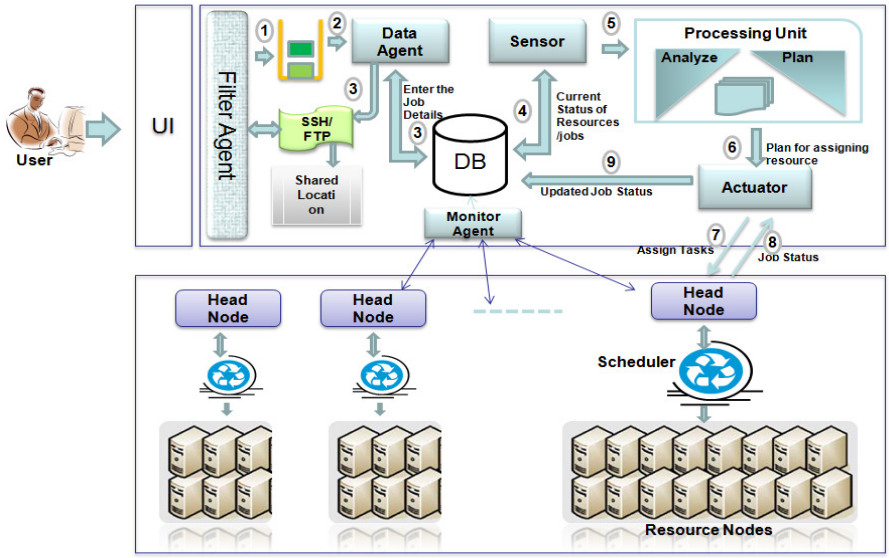

FIG. 3.6. *SPM Detail Working*

3.2. SPM Working. Figure: 3.6 describes the complete process behind the SPM working. This describes the complete process, starting from user's submitting the job from its machine, all the way through to the user receiving the result of submission. Figure: 3.7 discusses the basic flow in few cases. The basic process is as follows:

1. User submits the job through the user interface. UI passes the job request to the *filtering unit*, to scan for any malicious activities. After filtering, jobs are published into the queues.
2. *Data agent* keeps on monitoring the job queue for any new requests. Once discovered any new request, the fetcher in the data agent, fetches the job details and inserts the details into the database.
3. The job data is pushed into the JOB table. Along with this, the data needed by this job is transferred to the central location using FTP. Filtering unit plays its role again to scan the packets before downloading the data to shared location. This helps in avoiding frequent security checks.
4. Sensor monitors the JOB and RESOURCES database and passes the information to the processing unit. The RESOURCES table is updated by the *monitor agent*. This helps in reducing the failure rate caused by invalid old information about the resources.
5. *Processing unit* analyzes the information provided by sensor, and publishes the plan describing which jobs are going to be executed first, which job's need to be resubmitted, what is the current system cpu usage, etc. This plan is shared to Actuator by the processing unit.
6. The *Actuator*, based upon the plan, starts passing the jobs to the meta scheduler (not shown in figure).
7. *Meta scheduler* then schedules the jobs to different head nodes (GRAM nodes in case of globus).

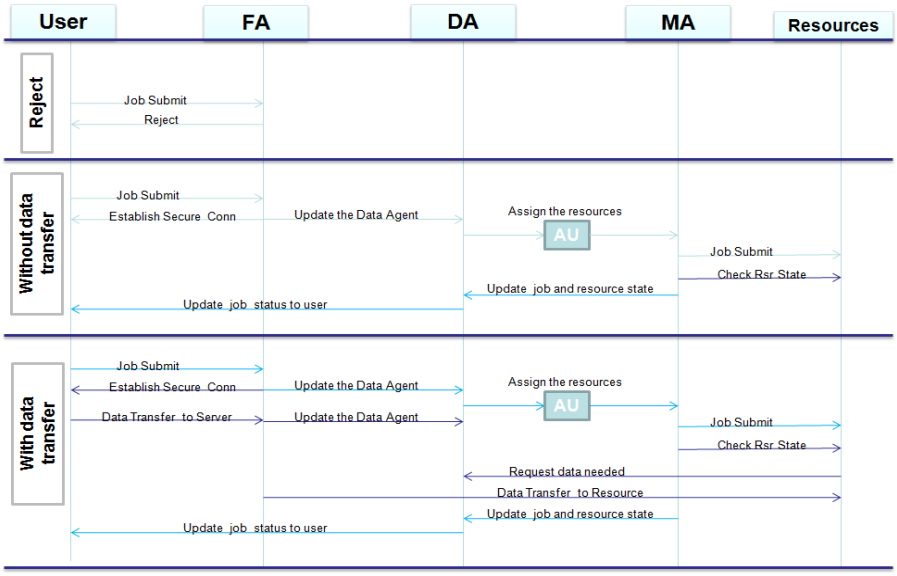

FIG. 3.7. *SPM Work Flow*

8. Actuator keeps monitoring the job status from the scheduler and updates the status into the JOB table.

4. Implementation and Results.

4.1. Softwares Used. The implementation uses: a standard IDS called Snort, ACID monitoring tool, apache web server to host ACID, MySql database and Java SDK6 for programming the tool.

4.2. Grid Environment. Grid Environment setup using Globus Toolkit 4 [37]. The major components involved are:

- *GRAM*: Enables resource allocation through job submission, staging of executable files, job monitoring and result gathering [35].
- *Scheduler*: Helps to schedule the jobs to resources available. Here we use the GridWay MetaScheduler [20] to schedule the jobs.
- *Grid FTP*: Extension of standard FT protocol that provides the secure, efficient and reliable data movements in grid environment [36]. In addition to standard FTP features, it provides GSI support for authenticated data transfer.

4.3. Experimental Setup. The SPM was implemented using the Snort. Snort is used to leverage existing distributed IDS capability to one step ahead by automating its rules generation process by the use of Genetic Algorithm. For testing the SPM in grid environment, we installed snort sensors on different resource nodes participating in the Grid Environmnet, then installed the database(MySql) on one machine, and deployed a Web server and ACID onto another machine. The grid environment is setup inside the TU campus using GT4 middleware. Grid environment consists of 44 Intel Dual Core 2.2GHz processor Windows XP nodes, 20 dual 2.4GHz Xeon Linux nodes and 5 node dual 450MHz PII Linux clusters. Each node has 1GB RAM and 80GB HDD. TU Grid is exposed to the outer world with limited access.

4.4. Experimental Results. To test the SPM, we setup the environment as described above. After environment setup, we configure some default rules in Snort which are later used by all nodes participating in the grid. The rules impose conditions on various GT4 components, these includes port check on GRAM (2119), MDS (2135), Grid FTP(2811), GSI SSH(22) over the TCP protocol for the default users. This rules database keep on increasing with time. As the time passed, the effectiveness of rules is decided on the basis of the alerts raised by it. If the rule raises the correct alert, its impact value is increased and vice-versa. Figure 4.1 is a graphical representation of the number of alerts raised with passage of time with default configurations.

We see that with the passage of time, security of the grid environment with SPM keep on increasing. As the time passed, the number of rules auto generated with help of genetic features- crossing over and mutation

FIG. 4.1. *Time vs Alerts raised*

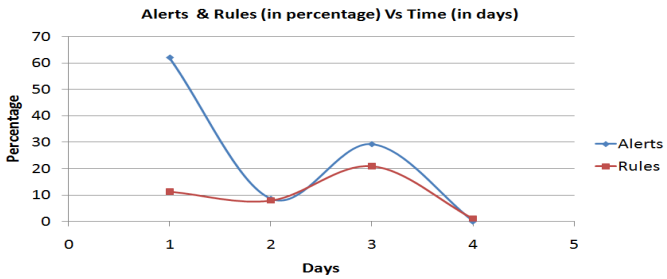

FIG. 4.2. *Alerts & Rules (in percentage) Vs Time (in days)*

also increases. We observed the SPM for the next few days after its setup in grid environment and found that based upon the alerts raised, rules keep on adding into the rule database (Figure 4.2).

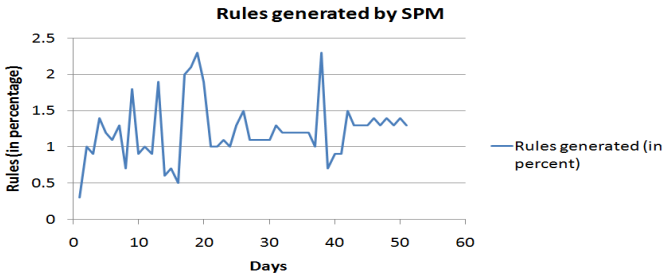

FIG. 4.3. *Rules generated by SPM*

As the rule database grow, more attacks are handled by the SPM. We have found that with respect to the manual system for configuring rules, our SPM reduces the attacks rate (Figure 4.4). After integrating with the current Grid environment, we have found that the attacks from the outside and inside world are reduces faster than the manual system (Figure 4.4). This is because of the increase in rules count due to the automated process implemented using the genetic algorithm.

5. Conclusions. As grid involves heterogeneous resources located in different geographical domains, secure and fault tolerant resource allocation services will increase its performance and efficiency. In this paper attempt has been made to list the most common security threats and the failures encountered by most of the present grid systems. The description of a new approach- SPM has been proposed to deal with failures and security threats with least manual administrative interference. This new approach has layered architecture comprising- Application Layer, Agent Layer and Autonomic Layer. SPM uses the Genetic Algorithm approach to provide the self-protection. The advantage of GA approach over existing approaches is that, SPM makes the system more robust. As central node transfers rules to all the child nodes once they are created through crossover or mutation, in case of central node failure, the system will still keep on protecting the local system. Another advantage is that this system is easily scalable. As the central node carries all the rules, once the new node is registered with the grid system, it will automatically receives all the existing rule set from the central node.

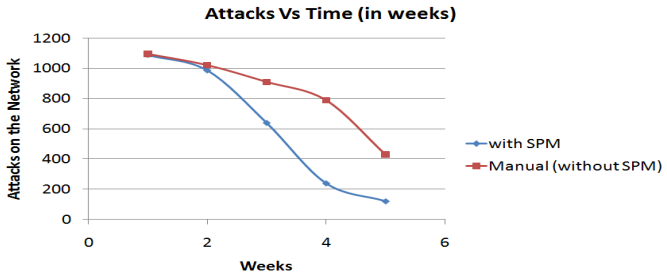

FIG. 4.4. *Attacks Vs Time (in weeks)*

The detailed working of SPM has been described. The goals achieved by the proposed system are summarized as:

- Reduction of administrative complexities because of decreased manual intervention.
- Automatic protection of the system from malicious activities like DoS, DDoS, etc by SPM's Self-Protection mechanism.
- Automatic monitoring of system performance.
- Scalability of the system to add new nodes easily into the grid.

With the implementation and results presented in the paper, we have tried to prove the prototype model for SPM. In the future work, there is the scope to test the feasibility and correctness of the automatically generated rules by GA.

SPM Algorithm

```
/*
Pass through the application layer
/
#VerifyUserData(){
        // validate the user
        IF user exist THEN
            IF PassThroughFilterAgent THEN
                Publish the job to queue
                JobId, UID, Priority, etc.
            ENDIF
        ENDIF
}
/*
filter agent working
/
#filterAgent(){
        GeneticRuleFormation();
        Transfer the rules to child nodes
}
#GeneticRuleFormation(){
        Initialization
            Based upon the existing rules, form the chromosome structure(hexadecimal format)
        Evaluation
            More the rule give the correct result, better is its fesibility factor
        WHILE termination criterion has not been reached
            Selection and Reproduction
                Based upon the feasibility factor select the set of chromosome
            Crossover
                two chromosomes are combined to reproduce the new offspring
            Mutation and Evaluation
                some major change is done to the existing chromosome to form the new chromosome
        ENDWHILE
        Update the rules database with new chromosome thus formed.
}
#monitorAgent(){
        Check the status of resources
        We will not be checking the resources actually, based
        upon the job statuses and time they are taking to execute
        we are predicting the grid load.
        IF NumberofJobspending/hr¿AvgNumberof Executing job/hr THEN
            Performance is degrading
        ENDIF
}
#dataAgent(){
        Collects the job info from queue
        IF job needs data from user THEN
```

```
            Form the ssh session
            Transfer the data to central location
        ENDIF
        Update the job information in the job's database
}
#raiseAlert(){
        At some time interval, agent will be called that will raise
        Alert for all the still cancelled jobs in the database
        Select job_id from job where state=cancelled
        IF count>0 THEN
            Raise the alert
        ENDIF
}
```

Acknowledgments. I thank Dr. Bhupinder Kaur, M.D.S. for clearing my doubts related to genetic algorithms.

REFERENCES

[1] Y. A. Mohamed and A. B. Abdulla, *Immune Inspired Framework for Securing Hybrid MANET*, ISIEA, IEEE (2009).
[2] Benoit Claudel and Noel De Palma and Renaud Lachaize and Daniel Hagimont, *Self-protection for Distributed Component-Based Applications*, Springer-Verlag Berlin Heidelberg (2006).
[3] Ramon Nou, Ferran Julia, and Jordi Torres, *The need for self-managed access nodes in grid environments*,IEEE, EASe (2007).
[4] M. Romberg, *The unicore grid infrastructure*, `http://www.unicore.org` (2002).
[5] B. Sotomayor and L. Childers, *Globus Toolkit 4: Programming Java Services*,(2005).
[6] Hang Guo, Ji Gao, Periyou Zhu, Fan Zhang, *A Self-Organized Model of Agent-Enabling Autonomic Computing for Grid Environmnet*, 6th Word Congress on Inteligent Control (2006).
[7] Alan Ganek, *Overview of Autonomic Computing: Origins,Evolution, Direction*,CRC Press(2004).
[8] Jeffrey O. Kephart and David M. Chess, *The Vision of Autonomic Computing*,IEEE Computer(2003).
[9] Marty Humphrey and Marry Thompson, *Security Implications of Typical Grid Computing Usage Scenarios*,Cluster Computing, Volume 5 , Issue 3(2002).
[10] Wei Li, *Using Genetic Algorithm for Network Intrusion Detection*,Proceedings of the United States Department of Energy Cyber Security Group (2004)
[11] Pohlheim Hartmut, *Genetic and Evolutionary Algorithms: Principles, Methods, and Algorithms*, Genetic and Evolutionary Algorithm Toolbox (2003)
[12] I. Chopra, M.Singh, *Analysing the need for autonomic behaviour in grid computing*, ICCAE P535-539, IEEE (Feb 2010)
[13] , *Stopping insider attacks: how organizations can protect their sensitive information*, ibm.com/services (2006)
[14] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke, *The physiology of the Grid*, Global Grid Forum(2002)
[15] Salim Hariri, Guangzhi Qu, and et.al., *Quality-of-Protection (QoP)-An Online Monitoring and Self-Protection Mechanism*, IEEE Journal on selected areas in communications, vol. 23, no. 10 (2005)
[16] P. Horn, *Autonomic computing: IBM's perspective on the state of information technology*, `http://www.research.ibm.com/autonomic/` (2001)
[17] Kevin J. Houle, George M. Weaver, *Trends in Denial of Service Attack Technology*, CERT® Coordination Center (October 2001)
[18] Ian Foster, Carl Kesselman, and Steven Tuecke, *The anatomy of the Grid*,John Wiley and Sons (2003)
[19] Ju Wang, Xin Liu and Andrew Chien, *Empirical Study of Tolerating Denial-of-Service Attacks with a Proxy Network*, Proceedings of the 14th conference on USENIX Security Symposium (2005)
[20] Ruben S. Montero, *The GridWay Meta-Scheduler,Open Source Grid and Cluster*, Oakland, CA (May 2008).
[21] Manish Parashar and Salim Hariri, *Autonomic Computing: An Overview*,Springer-Verlag Berlin Heidelberg (2005).
[22] Gerald Tesauro and David M. Chess and William E. Walsh and Rajarshi Das and et.al, *A Multi-Agent Systems Approach to Autonomic Computing*, AAMAS'04 ACM (Jul 2004).
[23] Christian Kreibich, Jon Crowcroft, *Honeycomb - Creating Intrusion Detection Signatures Using Honeypots*, ACM SIGCOMM, January (2004).
[24] Anirban Chakrabarti, *Grid Computing Security*, Springer, Ch-6 pp105, (2007).
[25] Snort, *https://edge.arubanetworks.com/article/leveraging-centralized-encryption-snort-part-1*
[26] Mark Stephen, V.S. Sukumaran Nair, Jacob. Abraham, *Distributed Computing Grids- Safety and Security, Security in Distributed, Grid, Mobile, and Pervasive Computing*, Chapter 14
[27] Anirban Chakrabarti, *Grid Computing Security*, Springer, Ch-8 pp159, (2007).
[28] S. Kenny and B. Coghlan, *Towards a Grid-Wide Intrusion Detection System*, Advances in Grid Computing. Springer, pp. 275–284, (2005).
[29] A. Schulter, F. Navarro, F. Koch, and C. Westphall, *Towards Gridbased Intrusion Detection*, Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, pp. 1–4, (2006).
[30] Michal Witold Jarmolkowicz, *A Grid-aware Intrusion Detection System, Technical University of Denmark*, IMM-THESIS (2007).
[31] Fang-Yie Leu, Ming-Chang Li, Jia-Chun Lin, *Intrusion Detection based on Grid*, ICCGI'06, (2006).
[32] M. Srinivas, Lalit M. Patnaik, *Genetic algorithms: A survey*, IEEE Computer, 27(6):17–26, June (1994).

[33] Michael P. Brennan, *Using Snort For a Distributed Intrusion Detection System*,SANS Institute (2002).
[34] Hassen Sallay, Khalid A. AlShalfan, Ouissem Ben Fred, *A scalable distributed IDS Architecture for High speed Networks*, IJCSNS, VOL.9 No.8, August (2009).
[35] M. Feller and I. Foster and S. Martin, *GT4 GRAM: A Functionality and Performance Study*, (2008).
[36] Data Management: Key Concepts, Grid FTP, `http:/www.globus.org/toolkit/docs/4.0/data/gridftp` (2008).
[37] Globus Alliance, *A Globus Primer: Describing Globus Toolkit Version 4*, `http:/www.globus.org/toolkit/docs/4.0/` (2008).

Edited by: Jemal H. Abawajy, Mukaddim Pathan, Al-Sakib Khan Pathan, and Mustafizur Rahman
Received: October 11th, 2010
Accepted: November 10th, 2010