An Autonomic Security Monitor for Distributed Operating Systems

Alvaro E. Arenas, Benjamin Aziz, Szymon Maj, and Brian Matthews

[1] Department of Information Systems, Instituto de Empresa Business School, Madrid, Spain
alvaro.arenas@ie.edu
[2] School of Computing, University of Portsmouth, Portsmouth, U.K.
benjamin.aziz@port.ac.uk
[3] AGH University of Science and Technology, Krakow, Poland
smaj@student.agh.edu.pl
[4] e-Science Centre, STFC Rutherford Appleton Laboratory, Oxfordshire, U.K.
brian.matthews@stfc.ac.uk

Abstract. This paper presents an autonomic system for the monitoring of security-relevant information in a Grid-based operating system. The system implements rule-based policies using Java Drools. Policies are capable of controlling the system environment based on changes in levels of CPU/memory usage, accesses to system resources, detection of abnormal behaviour such as DDos attacks.

1 Introduction

Monitoring is the act of collecting information concerning the characteristics and status of resources of interest. Monitoring open distributed systems is an active research area, and monitoring security properties is still considered a challenge. The aims of this paper is to present the monitoring of security-relevant information in the distributed operating system XtreemOS [5], a Grid-based operating system (OS) based on Linux.

The main contributions of the paper are the following. First, describing an abstract architecture for monitoring security properties in a distributed operating systems. Second, presenting an autonomic system that triggers corrective actions on monitored events. Finally, showing the implementation of the architecture and its integration into the XtreemOS operating system

The structure of the paper is as follows. Next section acts as background section, introducing the main concepts related to monitoring distributed systems. Section 3 focuses on the XtreemOS systems, describing its general monitoring subsystem. Then, section 4 explains how the general XtreemOS monitor was customised for monitoring security properties. Section 5 describes an autonomic rule-based system that exploits monitored data in order to take some actions. Section 6 shows the implementation of the secure monitoring subsystem. Then, section 7 compares our work with others. Finally, section 8 concludes the paper and highlights future work.

2 Background

We start by revising the main concepts and terminology related to monitoring, following the terminology defined in [9].

- An *entity* is any networked resource, which can be unique, having a considerable lifetime and general use. Typical entities are processors, memories, storage media, network links, applications and processes.
- An *event* is a collection of timestamped, typed data, associated with an entity, and represented in a specific structure.
- An *event schema*defines the typed structure and semantics of an event.
- A *sensor* is a process monitoring an entity and generating events.

Our interest is in monitoring distributed operating systems, and in particular Grid-based operating systems. Hence, we use as a reference the Grid Monitoring Architecture (GMA) [7] proposed by the Open Grid Forum. The main components of the GMA are the following.

- A *producer* is a process providing events.
- A *consumer* is any process that receives events
- A *registry* is a lookup service that allows producers to publish the event types they generate, and consumers to find out the events they are interested in.

After discovering each other through the registry, producers and consumers communicate directly. GMA defines three types of interactions between producers and consumers. Publish/subscribe refers to a three-phase inter-action consisting of a subscription for a specific event type, a stream of events from a producer to a consumer, and a termination of the subscription. Both the establishment and the termination of a subscription can be initiated by any of the two parties. A *query/response* is an one-off interaction initiated by a consumer and followed by a single producer response containing one or more events. Lastly, a *notification* can be sent by a producer to a consumer without any further interactions.

3 Monitoring in a Distributed Operating System

This section first presents a brief description of the XtreemOS distributed operating system and then gives a general overview of its monitoring component. The XtreemOS Grid OS is based on the Linux OS, extended as needed for enabling and facilitating Grid computing [5]. XtreemOS Grid spans multiple administrative domains on different sites, comprising heterogeneous resources that can be shared by the participating organisations. As illustrated in Figure 1, XtreemOS is composed of two subsystems:

- The XtreemOS foundation, called *XtreemOS-F*, is a modified Linux kernel embedding Virtual Organization (VO) support mechanisms and providing kernel level process checkpoint/restart functionalities.

– The high-level Grid services, called *XtreemOS-G*, which comprises several Grid OS distributed services to deal with resource and application management in VOs, and it is implemented on top of XtreemOS-F at user level. The three main subsystems of XtreemOS-G comprises: Data Management, which federates multiple data stores located in different adminitrative domains; VO Management, which manage the life-cycle of VO, including the management and enforcement of VO security policies; finally, the Application Execution Management (AEM) subsystem is in charge of discovering, selecting and allocating resources for job execution, as well as starting, controlling and *monitoring* jobs. XtreemOS also includes a set of services facilitating scalability, including a scalabe publish/subscribe subsystem.

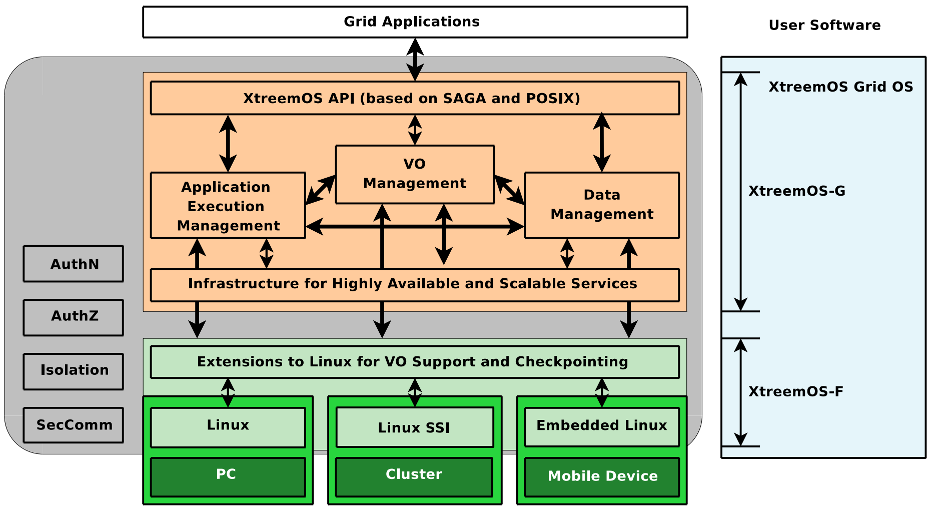
Grid Applications
User Software
XtreemOS API (based on SAGA and POSIX)
XtreemOS Grid OS
VO Management
Application Execution Management
Data Management
XtreemOS-G
AuthN
Infrastructure for Highly Available and Scalable Services
AuthZ
Extensions to Linux for VO Support and Checkpointing
Isolation
Linux
Linux SSI
Embedded Linux
XtreemOS-F
SecComm
PC
Cluster
Mobile Device

Fig. 1. XtreemOS Software Architecture [5]

XtreemOS follows the philosophy of associating a job with multiple processes running on several nodes, similar to the Linux process-thread paradigm. To this end, XtreemOS defines a hierarchy of entities composed of the job, job unit and process. In order to check the status of jobs and processes, XtreemOS AEM includes a monitoring infrastructure [6], which allows one to monitor the system with user-defined events at the abstraction levels of jobs, processes and threads.

Figure 2 shows an abstraction of the XtreemOS monitoring infrastructure. XtreemOS AEM includes two main components: the *Job Manager*, which provides job management features such as scheduling and storing job-related information; and the *Execution Manager*, which manages execution of jobs at the process level. Monitoring is performed at each of these levels, based on events defined by the user or by the system. XtreemOS events, called metrics in [6],

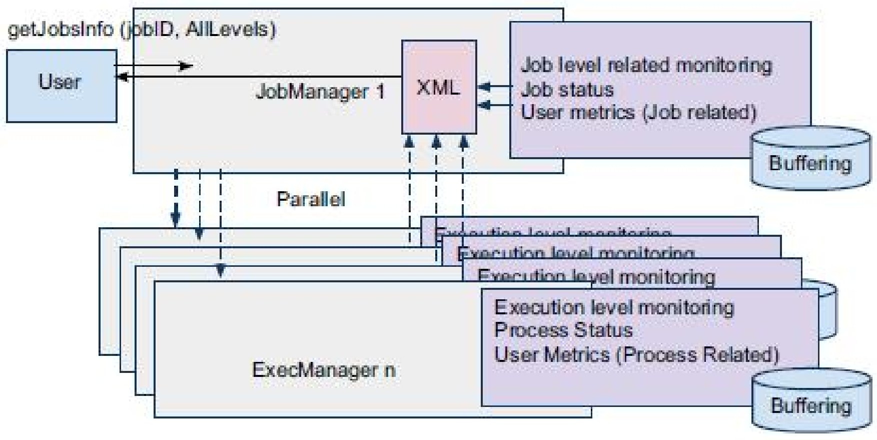
getJobsInfo (jobID, AllLevels)
User
JobManager 1
XML
Job level related monitoring
Job status
User metrics (Job related)
Buffering
Parallel
Execution level monitoring
Execution level monitoring
Execution level monitoring
Execution level monitoring
Process Status
User Metrics (Process Related)
ExecManager n
Buffering

Fig. 2. XtreemOS Monitoring Infrastructure [6]

have an event schema including a value type (Boolean, integer, timestamp, $\cdots$) and a scope. A scope (JOB, JOBUNIT or PROCESS) indicates the source of monitoring information.

The monitoring service in XtreemOS is a general one, used by AEM services as well as other XtreemOS services, such as the service for monitoring security described in the following section.

4 Monitoring Security in XtreemOS

Our main objective is to exploit the XtreemOS monitoring infrastructure in order to assess the security status of the distributed system. This section describes an instantiation of the XtreemOS monitor subsystem to monitor security events. First we describe the architectural distribution of the XtreemOS monitor; then, we introduce the main use cases exploiting the security-monitoring capabilities; and finally, outline the security events defined.

4.1 Security Monitoring and Auditing

XtreemOS defines two types of sites (organisations) participating in a distributed system: A *core site*, which is a site hosting and executing XtreemOS services and is considered essential; usually it includes services such as VO membership management, VO security policy management and auditing services. Any other site participating in a XtreemOS Grid is considered a *resource sites*, usually providing resources to the Grid.

We define two types of monitors. First, the *resource monitor* is responsible for monitoring resource-related events such as CPU utilizations, memory usage, network traffic, job status and job exit codes. There is one resource monitor for each site participating in a Grid. Second, the *core monitor* monitors events related to the Grid and VOs, and information collected from core XtreemOS services such as VO Management. There is one instance of the core monitor executing in the core site of the XtreemOS system.

In addition, we define an auditing service responsible for managing the monitored information. It uses the monitors as a source of information, which is stored in historical database for later analysis. There is one instance of the auditing service executing in the core site of the XtreemOS system.

4.2 Security Requirements

In order to define what type of information should be monitored, we analysed a portfolio of fourteen Grid applications and identified the following use cases in which the monitoring of security-related information is required:

- *CPU/Memory Usage Restrictions*. In this requirement, the system should be able to register the amount of computational resources used by a user's total number of jobs. As soon as the user's jobs use more than the specified quota, the system takes an action; for instance, to forbid submitting any more jobs by that user. A variation of this requirement would be to allow the quota to be dynamically calculated based on the status of the distributed resources, rather than presetting it statically during the initialization of the system. The restriction on user actions may have expiration time, which may also be based on predictions.
- *Defense Against DDos Attacks.* The aim of this case is to make the system more resistant to Distributed Denial of Service (DDOS) attacks. A resource analyzes incoming packets locally and exchanges aggregated meta information with other resources, hence somehow monitoring the traffic data. Accumulated knowledge enables detection and prevention of such attacks.
- *Dynamic Access Control.* Cases like this aim at restricting potentially harmful accesses to resources dynamically. An access to a resource is governed by dynamically changing attributes, which could be managed and controlled according to either a Separation of Duty model or a Chinese wall policy model.
- *VO Usage Policy Enforcement.* This case restricts available actions in a VO. Users actions in the VO are monitored and possibly stopped and/or logged if they do not match the predefined policies. These may forbid usage of certain actions or restrict the frequency of such actions. Monitoring repeated unreasonable requests or failed attempts to use the resource in a short amount of time may lead to the suspicion that the user is attempting to misuse the resource with ominous intent.
- *Malicious Behaviour Detection.* Another important requirement of Grid systems is to detect possible misbehavior of jobs in a Grid environment and

warn users in time to minimise the risk of damage from malicious behaviour. If the monitoring system detects certain characteristic patterns in events over some period of time during job executions, then this may imply execution of malware, therefore, requiring further the release of a warning to the affected users.

– *Peak Hours Detection.* This requirement is related to determining time periods during which the resources are scarce and times when resources are abundant. The system monitors resource availability in time and determines trends, presenting useful statistical information.

4.3 Monitoring and Auditing Capabilities

Based on the requirements, the XtreemOS system has defined the following monitoring and auditing capabilities.

Monitoring Capabilities. These are related to the monitoring of various information related to resource metrics, jobs, events, nodes and policy violations.

– *Monitoring Resource Metrics.* This capability allows the administrators to obtain notifications when particular values for resource metrics change or reach certain levels. For example, these include CPU utilization levels, memory usage levels and the amount of network traffic.
– *Monitoring Jobs.* This use case allows the administrators to monitor job-related information. Different job metrics can be monitored, for example, job status, job submission time and job exit status, as well as higher-level information such as the number of jobs currently running on a node or over several nodes.
– *Monitoring Nodes.* This includes the monitoring of various nodes in the Grid. Example of what can be monitored on a node includes its state and the state of the containers running on the node.
– *VO Policy Violation Monitoring.* This capability generates notifications about any policy violations in the system. In an autonomic policy system, this is very important as it may trigger the evolution process for new rules and policies. For example, if a user continuously violate their CPU usage quota on a particular node, it may trigger a new rule that blocks the specific user from submitting future jobs to the node.

Auditing Capabilities. Auditing capabilities include any functionality that is based on the information gathered from the monitoring capabilities. These include archiving and securing monitored data, querying historical database and the generation of the various VO, node and user behaviour reports.

– *Archiving and Securing Monitored Data.* This capability simply allows any monitored data to be archived in a historical database. In most cases, monitored data is sensitive information that needs protection for future references. Hence, this capability includes functionalities to protect monitored data by using encryption or access control mechanisms.

- *Querying Historical Database.* This use case allows the querying of the historical database in order to retrieve information about past events at different level of granularity: process, jobs, VOs.
- *Report Generation.* This capabilities allows generation of detailed reports about either the VO state, the state of a particular node in the Grid, or a specific user's behaviour over a period of time.

5 Autonomic Management of Security Events

The monitoring and auditing capabilities described previously allows one to monitor events, store them in a database, and query for particular events. In order to achieve more autonomous behaviour, we have extended the monitoring capabilities with a rule-based system able to analyse monitored information in real-time and take corrective actions accordingly, which themselves could lead to new rules.

The rule-based system consists of four elements: *Event Feeder*, which provides the stream of external events generated by the monitors into the autonomic system; *Rule Engine*, the logic of the system; *Rule Base*, which contains the collection of rule defined in the system; and *Action Executor*, which takes action to affect external environment. In addition, there is a *Working Memory* that is built dynamically from incoming facts and events during the life of system. These main components of the architecture are shown in Figure 3.

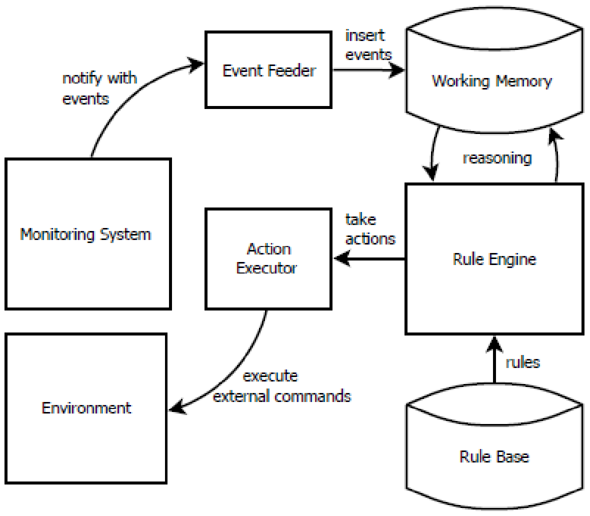

Fig. 3. Architecture of the XtreemOS Autonomic Monitoring Sub-System.

Next, we describe in more detail each of these components.

- *Event Feeder* is the part of architecture responsible for communicating with the monitoring system. Its implementations gather events either by subscribing to notifications from the monitoring system or by reading event objects from external streams, like files. Events are then inserted directly into the working memory, where the rule engine should react to them immediately. Different implementations of this subsystem can handle different monitoring systems and varying event formats, converting events into a suitable type if necessary.
- *Rule Engine* works in stream mode, which means it can analyze events in real-time, immediately firing any applicable rules, as opposed to filling agenda with activations and periodically firing rules in determined order. When a rule activated by some incoming event fires, the result may call the action executor as well as add new facts into the working memory. The newly added facts may lead to other rules being fired and this potentially-recursive process is called *reasoning* or *inferring*.
- *Rule Base* contains all of the system's logic except for the relation to the external environment. It is loaded into the system and compiled during initialisation, therefore it is not possible to modify it during runtime. Rules however can be modified before loading the system, but also may be configurable while being loaded into the system.
- *Action Executor* is responsible for the manipulation of external environment. Consequences of some rules may affect external environment, for example restricting user's accesses, as opposed to rules only affecting the working memory.

6 Implementation

The general XtreemOS monitoring component is part of the Application Execution Management (AEM) component [6]. XtreemOS monitoring includes events and metrics. Examples of events include "job failed", "VO created" or "user certificate not valid". By contrast, metrics are user-defined and has associated a value. Examples of metrics include "cpu utilization - 80%", "free disk - 100.5 GB" or "jobs running - 5".

Information for Process Monitoring is obtained from a daemon reading the `/proc/pid` file in the nodes. The Job Monitoring implementation provides interfaces to get the information associated to jobs (*getJobsInfo*, *getJobMetrics*); mechanisms to add new information to the generated by the system - user metrics (*addJobMetric*, *setMetricValue*, *removeJobMetric*); and mechanisms to be notified when certain monitoring events fire (*addMonitoringCallback*).

The Monitoring Manager collects monitoring data from various sources and stores it for a period of time. Interested parties define monitoring rules which describe what to monitor. When conditions of the monitoring rule are met, a notification is issued. A particular monitoring rule is indentified by monitoring rule name to which interested parties subscribe to receive notifications. The Monitoring Manager implementation provides interfaces for saving events and metrics (*saveEvent*, *saveMetric*); mechanisms for setting monitoring rules and subscribing to monitoring notifications (*addNotification*, *subscribe*, *unsubscribe*); and functionalities for defining aggregated metrics.

The Auditing Manager permanently stores monitoring data received from the Monitoring Manager. Data is archived in a history database that can be later analyzed and used for generating reports. The Auditing Management implementation provides interfaces for defining archiving rules (*addArchiveRule*, *cancelArchiveRule*); and querying the database (*query*). Hibernate is used as query language.

The implementation of the Autonomic Manager of Security Events was carried out using the Java Drools technology[5], a Java-based platform for developing writing rules, workflows and performing event processing. We have defined a Manager class that encapsulates and hides Java Drools interface in order to simplify starting and stopping of the system. The Drools rule engine itself is not thread-safe, but Manager synchronizes all necessary methods, thus the whole system may be used by multiple threads, as its methods are non-blocking. The Manager implements the *EventEntryPoint* interface, which is passed on as an argument to the EventFeeder when it starts. In additiona, we have defined a Configuration call that sets the initial parameters of the system. Objects representing parameters are inserted into the working memory when starting the system, therefore any subsequent additions will not affect execution.

7 Discussion

The approach we followed in this work on monitoring security information is multi-layered. The first layer is security agnostic, i.e. low-level information is detected using the XtreemOS AEM infrastructure [6], monitoring the states of various processes and jobs. Second, based on the information collected by the AEM, a security-aware monitoring and auditing service [8] is implemented, whose monitored events could be queried directly from a database. Finally, an autonomic security monitoring service is also implemented based on the information collected from the AEM monitor; the service is dynamic in the sense that is able to evolve the various rules depending on the status of resources, the jobs running within, and the environment.

This is not the first attempt at achieving comprehensive monitoring in Grid systems. In [7], the authors define a full Grid monitoring architecture, though the architecture is designed with performance of Grid systems in mind, rather than security. In [4], the authors monitor security information in Grids, which is used only for brokering and selecting resources.

There are many systems that have been developed to support monitoring in Grids, however, Ganglia [3] one of the most widely used such systems within the Globus community, as a result of its integration with the Globus Meta Directory Service (MDS) [2]. In [1], a Grid monitoring infrastructure is defined,

[5] http://www.jboss.org/drools

called OCM-G, which can be used to support the development of various Grid monitors.

8 Conclusion and Future Work

This paper describes the monitoring of security properties in the XtreemOS operating system. Monitoring security is seen as a particular case of XtreemOS monitoring, where relevant events and user metrics are monitored and aggregated in order to determine potential security problems. In addition, we presented a Java-Drools-based autonomic monitoring system, which further extends the functionality of the standard security monitoring service in XtreemOS with capabilities for the evolution of rules and policies based on the dynamic information collected from the resources, jobs and VOs.

There are many directions for future work. Mainly, we would like to exploit the XtreemOS autonomic monitoring service for the enforcement of more complex autonomic security policies, in particular, focusing on the runtime detection of malicious job signatures that could imply viral behaviour. The autonomic monitoring service itself is somehow independant of the XtreemOS system in that it only relies on the information collected by the AEM, therefore, another main direction for future work will involve integrating the service with other Grid middleware systems, in particular Globus and gLite.

References

1. Balis, B., Bubak, M., Funika, W., Szepieniec, T., Wismuller, R., Radeck, M.: Monitoring Grid Applications with Grid-Enabled OMIS Monitor. In: Proc. First European Across Grids Conference. pp. 230–239 (2003)
2. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. In: 10th IEEE International Symposium on High-Performance Distributed Computing (2001)
3. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing 30(7), 817–840 (2004)
4. Mazzoleni, P., Crispo, B., Sivasubramanian, S., Bertino, E.: Efficient Integration of Fine-Grained Access Control and Resource Brokering in Grid. The Journal of Supercomputing 49(1), 108–126 (2009)
5. Morin, C., Jégou, Y., Gallard, J., Riteau, P.: Clouds: a new playground for the xtreemos grid operating system. Parallel Processing Letters 19(3), 435–449 (2009)
6. Nou, R., Giralt, J., Corbalan, J., Tejedor, E., Fito, J.O., Perez, J.M., Cortes, T.: XtreemOS Application Execution Management: A Scalable Approach. In: 11th ACM/IEEE International Conference on Grid Computing (2010)
7. Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V., Wolski, R.: A Grid Monitoring Architecture (2002)
8. XtreemOS Consortium: Fourth specification, design and architecture of the security and vo management services. In: XtreemOS public deliverables - D3.5.13. Work Package 3.5 (Dec 2009), `http://www.xtreemos.org/publications/public-deliverables/`
9. Zanikolas, S., Sakellariou, R.: A Taxonomy of Grid Monitoring Systems. Future Generation Comp. Syst. 21(1), 163–188 (2005)