Grid Security: The host approach

Evgueni Dodonov, Jessica Antonieta Zorzatto, Joelle Quaini Sousa, Hélio Crestana Guardia

Departamento de Computação, UFSCar
Rodovia Washington Luis, km. 235
São Carlos (SP), Brazil

(eugeni, jessica, joelle, helio}@dc.ufscar.br

Abstract: *Security is an important concern in providing the infrastructure for the implementation of general purpose computational grids. However, most grid implementations focus their security concerns in correctly authenticating users and hosts and in the communications among them. In most cases, application security is left to the underlying operating system. This can be a problem when a "malicious" application is executed. In this work, we introduce the GridBox architecture, that aims to provide additional security for GRID applications, using Access Control Lists and sandbox functionality for GRID tasks.*

1. Introduction

The infrastructure of a Computational Grid should supply basic services for control, communications and secure sharing among the Grid's resources. Because this computational structure is highly distributed, shared, and sometimes interlinked through a public data communications channel, security is also a crucial issue.

In order to provide security for the GRID *communication*, some sort of connection security, such as *SSL*, *GSS-API* and *X.509 certificates* is used. However, most of the existing GRID architectures do not provide additional security for the GRID *applications*, making it possible to harm the members of a GRID network with malicious code.

In this paper, we present an architecture for computational GRID security, created to provide additional security for GRID applications. The architecture uses the concepts of *sandbox* and *Access Control Lists* in order to limit the functionality of grid applications, allowing a fine tuning of most system accesses by every application.

The mapping of the resources made available by the hosts and the requirements of applications is achieved using a web portal for the ProGrid Environment.

The paper is structured as follows: a series of different security approaches is shown in the section 2. Together with a discussion about existent grid architectures and their security models. The proposed *GRIDBOX* architecture is described in section 3. Finally, the section 4 presents the results obtained and conclude this work.

2. GRID Security

The traditional UNIX security model (*Discretionary Access Controls* - ***DAC***), composed of a limited set of permissions (namely, user-group-other permissions) does not provide the flexibility necessary for modern systems, making the security of

processes a crucial point of overall system security. This security model does not provide additional security for different processes, allowing a single process error to affect other processes, especially when run as a privileged user.

Secure programs have to minimize privileges so that any malicious programs are less likely to become security vulnerabilities. Even programs formally proven correct using sophisticated mathematical techniques are prone to have malfunctions. The three main approaches to security enforcement are:

• **Process separation** – this approach consist of separation of processes in the system, usually by limiting the file system and memory accesses (*chroot*, *sandbox* and *jail* approaches).

• **Security enforcement** – this approach attempts to tighten the security by defining all possible process functions and allowing limited access to the system.

• **System restriction** – *UMLinux* [Dike 2001] and *VServer* [VSERVER 2004] introduced a new security model, which consist of the creation of different *virtual* servers in the same operating system. This way, each process has the usual access to its system but runs independently from each other.

Regarding file access permissions, the **Access Control List** (**ACL**) solution associates a *list* of *permissions* for an object, extending the traditional *Unix* file permissions. This way, it is possible to define precisely which users may access each file. This solution is being used in the *NTFS* file system [NTFS 2004] and in the Linux kernel, using Extended Attributes technology [Grünbacher 2004].

At the Operating System level, **SELinux**, the Security-enhanced Linux [SMALLEY 2002], based on the LSM (Linux Security Modules) framework, does not use the *DAC* mechanisms which determine what a program can do based only on the identity of the user running the program and ownership of objects such as files, sockets and so on. Instead, SELinux is based on *MAC* (Mandatory Access Control) mechanisms, which allows complete control over each and every system component. However, the main drawbacks of the SELinux technology are the need to define precisely all rules for its *policy* and a complex policy design.

UMLinux, the *User Mode Linux* solution aims to provide a complete linux-inside-linux solution for any kind of application. User-Mode Linux creates a virtual machine that may have more hardware and software virtual resources than the actual, physical computer. This approach provides a good security at the cost of running an operating system on top of another, thus decreasing the performance of all operations.

The **Sandbox** [Gong 1997] approach is a light-weighted model of process separation. This approach is widely used in the JAVA Architecture, and provides a reserved area for each application, the sandbox, inside which the application can perform its operations normally, restricting the access to the outside objects. This approach combines good security with a low overhead, thus offering good performance. However, in most cases this approach is not implemented on the system level.

The **chroot** [Mcgrath 2004] command and system call can be considered a reduced sandbox, as it only restricts the filesystem operations of a process. Furthermore, it is possible for the applications run as the system administrator to escape the *chrooted* environment and gain full file system access [Simes 2004].

The **Jail** [Kamp 2000] approach is a FreeBSD-only implementation of a combination of chroot with sandbox approach. Basically, each *jailed* application has its own space together with an IP address. In this way, the security of the *sandbox* approach is implemented directly into operating system, offering good performance combined with tight security.

As can be seen, different approaches can be used in order to enforce the basic security provided by the operating system. Some of these approaches are restricted to the operating system used (such as *jail*, which can only be used on FreeBSD systems or *UMLinux*, only available on Linux machines), which is not the case for a GRID network. In a grid network, different operating systems and machine architectures may be present, thus limiting the availability of some of the present security technologies.

Regarding the development of grid applications, known grid architectures include the Globus Toolkit [GLOBUS 2004], Apple XGrid [XGRID], OurGrid [Andrade 2003] and the ProGrid technologies [Costa 2003].

The **Globus** Security Infrastructure (GSI) enables a decentralized security management together with a *single sign-on* for users of the grid. It is based on X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol. GSI also supports proxy credentials, inter-operability with local security mechanisms, local control over access, and delegation.

Security within the **Xgrid** architecture is enforced by the default UNIX permission model, where the daemons are running under an unprivileged user's permissions which has limited access to the system. Also, a password is requested by default for connections, further increasing the security. Currently, only the password negotiation is encrypted, while transmitted data is unencrypted.

The **OurGrid** project provides support for grid economy, disallowing the abuse of grid resources by greedy applications together with a security mechanisms for authentication, authorization and similar. However, as far as we could observe no additional application security is provided, regarding a malicious application submited for execution.

The **ProGrid** approach introduces the concept of *Grid Proxies*, which intermediates the communications made by the hosts at each site. The proxy servers aim to increase the security of inter-grid communication providing support for host and user authentication and security for messages exchanged between different grid networks.

The main objective of the ProGrid project is to provide transparent proxies for different grid subnetworks, acting as *gateways* between these networks. All communications between different networks require user and application authentication, increasing the overall grid security.

The main goal of any Grid Portal environment is its simplicity presented in an intuitive interface, which enables the access to the underlying system. Gaining access through the authorization and authentication interface, a client script for remote host configuration that provides its allocation rights must be loaded. The default policies necessary to execute remote tasks such as code compiling and running, performing process execution and job operations (such as job submission and deletion), jobs status control and jobs history are enabled by the use of scripts. Further rules can be configured for a safer application execution by using the GridBox tool.

However, as the ProGrid approach interacts between different grid subnetworks, the security of individual hosts is not enforced. The ProGrid aims to provide secure connections between hosts and networks but not preventing malicious application to compromise the grid network.

Currently, none of the studied grid architectures focus on application security, leaving most security aspects to the operating system. In the time of the first grid projects, such security aspects had no practical aspects, as each grid project had complete control over the data transmitted. This way, a malformed data could hardly compromise the grid network. The security, however, becomes a significant concern in modern general purpose grid infrastructures, where a malicious code can damage the grid.

3. GRIDBOX Architecture

The *GRIDBOX* architecture addresses grid security at the application level. While most GRID architectures provide security services for communications and authentication, the security of applications is usually left to the underlying operating system. This can be a problem, especially as the number of grid nodes increases.

Further, a security architecture is required to protect the computational resources of users willing to offer their machines for the execution of applications in a GRID.

In order to provide additional security services for grid applications, the *GridBox* architecture was conceived. The main objectives of this architecture are:

• **System security** – the system must be protected from possible hostile acts performed by applications

• **Interoperability** – concurrent applications must not be allowed to interfere with each other. E.g., application *A* must not modify application *B*'s files, and so on.

• **Tunable settings** – each application can have different requirements for system functions. For example, application *A* must have access to the network subsystem but not to the filesystem; and application *B* only uses the filesystem.

• **Transparency** – applications should not have to be concerned about the underlying security subsystem. System calls should occur as usual, and the security subsystem must take care of each possible violation.

• **Isolation** – each application must run independently from other applications on the same machine.

In order to accomplish all these requirements, the so-called *process limitation* was implemented in the *GridBox* architecture. In this sense, the GridBox architecture intercepts the system calls made by the applications and checks them against a pre-defined **access control list** (**ACL**). If the request is explicitly defined in the ACL, the action may proceed, either effectively executing the request or returning an error code.

The ACL is created just before the process execution, based on:

• **Node profile** – each grid node has its profile of execution, specifying basically which actions are permitted. A Node Profile defines only the default permissions for each subsystem. It is possible to detail the actions on a per process basis.

• **Application policy** – in order to further define the application behavior, each application may have an application policy file, which describes each action that the application must be allowed to perform.

All these limits are independent from the application being executed, and are being enforced transparently by the GridBox.

3.1 Implementation

In order to transparently enable the *GridBox* interaction with existing application the architecture was implemented as a *shared library*. The library is preloaded into memory before real application execution, overriding the system provided functions. The implemented *GridBox* library can be used on any unix-like operating system which supports shared library technology. Statically linked applications can also run in a protected manner if compiled with GridBox security enabled.

3.2 Application Deployment and Execution

Considering the bag of tasks model of grid implementation, the security model operates as follows:

• A user adds a host to the grid specifying its basic restrictions when accepting applications from remote hosts. A very restrictive profile can be used as default. Otherwise, one can determine which directories and files can be used, as well as CPU limitations, and communication (sockets) restrictions.

• A special local user account is created at each host for use with remote applications.

• Grid applications are deployed to the specified (or default) hosts on which they will be executed.

• The new shared library is preloaded into the memory prior to the activation of a grid application. As the *GridBox* library is loaded into memory, it reads the configuration file and processes the *ACLs* defined.

• The application is initiated being limited at startup by the CPU restrictions imposed at each host. During run time the code is also checked against the local access restrictions.

After the library is loaded into memory and the *ACL* is fully processed, the application is executed. Every appropriate system call is then intercepted by the *GridBox* at run time and verified before its real execution.

4. Conclusion and future work

In this paper, we presented an architecture for GRID security, composed of *sandbox* controlled by a set of *access control lists* for grid applications.

This approach allows the security of grid applications to be tightened, restricting their access to the system resources. Thus, the possibility of a malicious code execution is significantly reduced, disallowing grid application from damaging the host system and interfering with other applications on the same host. Using the proposed architecture, different aspects of every grid application can be controlled, allowing and disallowing system calls based on the defined *ACLs*.

Considering the performance interference introduced in an application, it depends on the number of system calls realized. On the other hand, the proposed model increases the user confidence when accepting applications to be executed on a general purpose grid implementation.

For future work, the project of an analyzer tool for automatic ACL creation for grid applications is being conducted. Beside that, architecture for the ProGrid environment, featuring automatic web-based grid application management is being developed. Our goal for that architecture is to provide automatic ACL generation for all grid applications, and to deploy the required GridBox library with the corresponding ACL to the grid nodes transparently.

References

[Andrade 2003] Nazareno Andrade, Walfredo Cirne, Francisco Brasileiro, and Paulo Roisenberg. OurGrid: An approach to easily assemble grids with equitable resource sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2003.

[XGRID] Apple Computer, Inc. X-grid project. http://www.apple.com/acg/xgrid/.

[Costa 2003] Paulo Costa, Sérgio Donizetti Zorzo, and Hélio Crestana Guardia. Progrid: A proxy-based architecture for grid operation and management. In *Proceedings of SBAC-PAD 2003*, 2003.

[Dike 2001] Jeff Dike. A user-mode port of the linux kernel, 2001.

[Gong 1997] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers. Going beyond the sandbox: An overview of the new security architecture in the Java Development Kit 1.2. In *USENIX Symposium on Internet Technologies and Systems*, pages 103–112, Monterey, CA, 1997.

[Grünbacher 2004] Andreas Grünbacher. Linux extended attributes and acls. http://acl.bestbits.at/, 2004.

[Kamp 2000] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. http: //docs.freebsd.org/44doc/papers/jail/jail.html, 2000.

[Mcgrath 2004] Roland McGrath and Free Software Foundation. Chroot – run command or interactive shell with special root directory. Linux Manual Pages, 2004.

[NTFS 2004] NTFS.COM. Ntfs - new technology file system designed for windows nt, 2000, xp. http: //www.ntfs.com/, 2004.

[Simes 2004] Simes. Breaking out of a chroot() padded cell. http: //www.bpfh.net/simes/computing/chroot-break.html, 2004.

[SMALLEY 2002] S. Smalley, C. Vance, and W. Salamon. Implementing SELinux as a Linux security module. NAI Labs Report #01-043, NAI Labs, Dec 2001.

[GLOBUS 2004] The Globus Project. http: //www.globus.org, 2004.

[VSERVER 2004] The Linux VServer Project. Linux vserver project. http: //www.linux-vserver.org/, 2004.