Um portifolio de segurança para um sistema de grade entre pares de livre entrada

Flavio Figueiredo, Matheus Gaudêncio, Thiago Emmanuel, Rodrigo Miranda, Francisco Brasileiro

[1]Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Laboratório de Sistemas Distribuídos
Av. Aprígio Veloso, s/n, Bloco CO
58.109-970, Campina Grande - PB, Brazil

{flaviov,matheusgr,thiagoepdc,vilar,fubica}@lsd.ufcg.edu.br

Abstract. *Security is an important aspect in grid computing, due to the necessity to protect the grid resources, users and middleware. In free-to-join peer-to-peer grids, the security issues are more complex, because there are not strong identities in the system. In fact, there is not a "one size fits all" solution for the users of this kind of grid. This paper adapts the security technologies available in the other kinds of grids and distributed systems, applying then to the free-to-join peer-to-peer grids, in order to build a security portfolio for the distinct user types in this kind of system. The portfolio was developed and validated on OurGrid.*

Resumo. *Segurança é um aspecto importante na computação em grade, pois é preciso proteger os recursos, os usuários e o próprio middleware da grade. Nas grades entre pares e de livre entrada, as questões de segurança se tornam mais complexas, devido à ausência de identidades fortes no sistema. Na realidade, não existe uma solução única que satisfaça os requisitos de todos os usuários deste tipo de grade. Neste artigo, as tecnologias de segurança disponíveis para os outros tipos de grades e sistemas distribuídos são adaptadas e aplicadas às grades entre pares e de livre entrada, a fim de formar um portifolio de segurança disponível para os diferentes perfis de usuários deste tipo de sistema. O portifolio foi implementado e validado no OurGrid.*

1. Introdução

É cada vez mais comum o uso de grades computacionais para o auxílio de pesquisas científicas que exijam um grande poder computacional. Há na literatura propostas de diversos modelos de grades, cada um apresentando requisitos distintos de segurança para atender às necessidades dos seus respectivos usuários. Em geral, existem dois problemas de segurança em grades: (i) a segurança dos usuários da grade contra recursos maliciosos; e (ii) a segurança dos recursos da grade contra usuários maliciosos. Uma forma comum de garantir a proteção de ambas as partes é fazer uso de um esquema de segurança baseado na confiança mútua entre os participantes [Foster et al. 1998].

O OurGrid é um *middleware* de grades computacionais que permite que qualquer usuário possa participar do sistema. Em `http://status.ourgrid.org` pode-se

ver um *snapshot* atual do sistema. Uma grade que use o *middleware* OurGrid é caracterizada como uma rede de livre entrada. Neste tipo de grade, onde não há garantia sobre quem são os participantes do sistema, soluções de segurança baseadas em confiança mútua se tornam inviáveis. Consequentemente existe a necessidade de construir mecanismos explícitos para dar suporte à proteção de recursos e usuários da grade.

Trabalhos anteriores já foram desenvolvidos para implementar segurança no OurGrid. A solução SWAN [Cavalcanti et al. 2006] provê ambientes de execução seguros, a fim de proteger os recursos da grade contra usuários maliciosos. Também existe uma solução para a proteção de usuários contra recursos maliciosos [Oliveira and Brasileiro 2006]. Embora eficazes do ponto de vista da segurança provida, tais soluções não são adequadas para todos os perfis de usuários que fazem parte da grade. O SWAN, por exemplo, demanda um grande esforço de implantação, o que inviabiliza a utilização da grade para alguns usuários mais leigos, enquanto que a solução de proteção contra recursos maliciosos consome muito poder computacional para executar réplicas de segurança das tarefas da grade.

Para disseminar a comunidade OurGrid, implementamos um portifolio de segurança que visa suprir diferentes demandas de diferentes perfis de usuários. O portifolio permite que usuários adotem a melhor solução de segurança para o seu domínio administrativo, levando em consideração questões de eficácia e facilidade de administração.

Na seqüência deste artigo, a Seção 2 expõe a arquitetura do OurGrid e a Seção 3 contextualiza os aspectos de segurança da Computação Voluntária e da computação em Grade. O cerne do artigo está na Seção 4, que mostra em detalhes o portifolio de segurança proposto, discutindo a sua implementação. Por fim, a conclusão enumera as contribuições deste artigo e os trabalhos futuros.

2. OurGrid

O OurGrid é uma grade de livre entrada, onde os participantes compartilham recursos ociosos de processamento e armazenamento de dados, para a execução de tarefas não comunicantes (*BoT*, do inglês *Bag of Tasks*). Uma característica buscada pelo OurGrid é fornecer um sistema facilmente implantável e com componentes bem desacoplados.

A arquitetura do OurGrid, como representada na Figura 1, é composta por quatro componentes principais: o Broker, que é uma interface cliente de submissão de tarefas; o Peer, entidade que agrupa consumidores e recursos de um domínio, controlando a doação e requisição de máquinas; o Worker, recurso executor das tarefas repassadas pelo usuário; e o DiscoveryService, um serviço de descoberta de recursos. Todos os componentes se comunicam utilizando o middleware JIC(*Java Internet Communication*) [Lima et al. 2006].

Um usuário que deseja executar um conjunto de tarefas no OurGrid deve usar o Broker para a submissão deste trabalho. O Broker então pede máquinas ao Peer de seu domínio (Peer local) que deve: fornecer todos os seus Workers disponíveis para a execução desta tarefa ao mesmo tempo em que, através do DiscoveryService, procura Peers remotos que possuam recursos apropriados. O Peer local então pede aos Peers remotos máquinas para o usuário do Broker. Os workers serão doados de acordo com um mecanismo de incentivo, denominado Rede de Favores, (*NoF*, do inglês *Network-of-Favors*), no qual o Peer remoto tenta entregar mais máquinas aos Peers que, no passado,

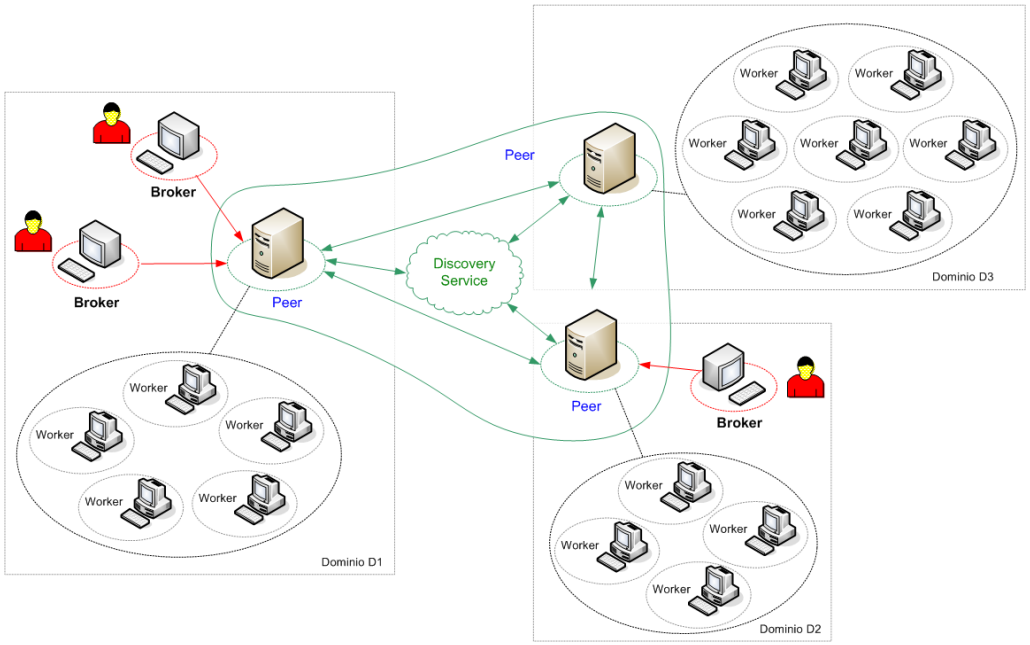

Figura 1. Componentes do OurGrid

lhe doaram mais recursos [Andrade et al. 2007]. O Broker, após receber cada recurso, escalona uma tarefa a ser executada imediatamente no Worker doado.

3. Contextualização

O OurGrid é um sistema de grade que tem características em comum com outros sistemas de computação distribuída. Para desenvolver um portifolio de segurança, inicialmente fizemos um levantamento bibliográfico sobre os diferentes modelos de computação distribuída e como estes provêem segurança para seus usuários.

3.1. Computação Voluntária

Computação voluntária é uma forma de computação distribuída em que recursos computacionais são disponibilizados espontaneamente por seus donos através da Internet. Sistemas que fazem uso deste modelo de distribuição alcançam taxas de processamento equiparáveis aos maiores supercomputadores construídos, como pode ser visto comparando projetos como o BOINC [Boinc Status 2007] e o ranking dos maiores supercomputadores do mundo [TOP 500 2007].

3.1.1. Segurança dos recursos

Para facilitar o ingresso de voluntários no sistema, devem ser fornecidas condições de segurança que satisfaçam as suas necessidades. Uma alternativa comum é controlar o processo de submissão das aplicações, que só ficam disponíveis para execução após serem verificadas como confiáveis pelo administrador da plataforma [Boinc Project 2007, Distributed.net 2007]. Geralmente, esta verificação implica na reescrita do código, o que diminui a facilidade de implantação de novas aplicações no sistema. Além disto,

é necessário que os donos dos recursos doados confiem em quem está realizando a verificação.

Uma estratégia bastante disseminada para a proteção de recursos contra aplicações é o uso de ambientes isolados, denominados de *sandboxes*. Implementações destes mecanismos incluem máquinas virtuais, jaulas no sistema operacional [Kamp and Watson 2000], modificações na visão do sistema de arquivos (*chroot*) e linguagens interpretadas [Gosling et al. 2005]. De forma geral, estes mecanismos mantêm sob controle um subconjunto seguro dos recursos que podem ser utilizados por processos não confiáveis.

Os sistemas para computação voluntária Bayanihan [Sarmenta 1998], Javelin [Neary et al. 1999] e o Unicorn [Ong et al. 2002] fazem uso de *Applets Java* como sandbox para as aplicações [Sun Microsystems 2007]. Esta arquitetura apresenta vantagens como a independência de plataforma e a baixa sobrecarga de configuração. Porém, as aplicações suportadas são restritas à linguagem Java.

O XtremWeb [Cappello et al. 2005] é um sistema entre-pares de computação voluntária, que implementa o isolamento através da *sandboxing* a nível do sistema operacional. No caso do XtremWeb, a limitação do que é permitido ser feito é definida através de políticas de acesso que controlam a manipulação do sistema de arquivos, conexões à rede e sinalização entre processos. Esta técnica é limitada, pois necessita de sistemas operacionais específicos e modificados para este fim.

3.1.2. Segurança das aplicações

Quando não se pode certificar que os voluntários são confiáveis, também é necessário proteger as aplicações contra eventuais sabotagens. Este tipo de falta pode ser tolerada usando técnicas de replicação e execução de tarefas com resultados conhecidos [Sarmenta 2001, Oliveira and Brasileiro 2006], o que torna possível a detecção de recursos maliciosos. Entretanto implica numa perda de processamento útil, proporcional ao grau de replicação.

Outra abordagem possível é a inclusão de marcações no código da aplicação. Esta técnica torna possível checar a integridade do código contra ataques em que recursos modificam a aplicação para retornar resultados inconsistentes. É possível para um usuário malicioso burlar esta técnica atráves de engenharia reversa do código para detecção das marcações. Alguns mecanismos como criptografia e obfuscação [Collberg and Thomborson 2002] se propõem a resolver este problema, dificultando o processo de engenharia reversa. Em grades computacionais entre-pares, diferente de computação voluntária, as aplicações em execução em um nó variam bastante, assim o trabalho de um sabotador em realizar a engenharia reversa não compensa o ganho que ele teria em sabotar a tarefa.

3.2. Computação em Grade

Os domínios administrativos que compõem uma grade possuem políticas locais de segurança. Uma solução de segurança que atua na grade idealmente não deve conflitar com as soluções locais de cada domínio. Outra preocupação é que relações de

confiança entre os participantes da grade não podem ser estabelecidas antes da execução das aplicações devido à natureza dinâmica da grade, em resumo não é possível saber a priori as entidades com as quais um relacionamento será estabelecido.

Foster et al. [Foster et al. 1998] descrevem, no contexto do projeto Globus [Foster and Kesselman 1998], uma arquitetura de segurança baseada em autenticação (ato de assegurar que uma entidade é quem diz ser) e autorização (comprovação do que é permitido ser feito por determinada entidade). Uma característica fundamental desta arquitetura é a independência de implementação, por exemplo, políticas de segurança podem usar qualquer tecnologia baseada em troca de chaves criptográficas. Além do Globus, outros sistemas como PBS [OpenPBS 2007], Sun Grid Engine [Engine 2007], TeraGrid [TeraGrid 2007] e gLite [gLite 2007], permitem apenas usuários autenticados e autorizados.

O projeto PUNCH [Kapadia et al. 2000] é uma plataforma de computação em grade que implementa uma solução de segurança que inclui: um *shell* modificado para aplicações interativas e o monitoramento em tempo de execução das chamadas ao sistema. Este *shell* modificado verifica se os comandos são seguros via políticas baseadas em controle de acesso e descritas através de um arquivo de configuração. Esta abordagem é útil quando utilizada para usuários específicos e aplicações não-arbitrárias, caso contrário a definição de diretivas de controle de acesso pode ser muito restritiva ou mesmo ineficaz desde que não se sabe o que estará em execução.

O Condor [Litzkow et al. 1988] é um sistema de gerenciamento de tarefas para computação intensiva. Este projeto evoluiu de forma que pode ser usado para construir uma infra-estrutura de grade, compartilhando recursos entre diferentes domínios administrativos [Frey et al. 2002]. A solução de segurança no Condor [Thain et al. 2005] incorpora: o gerenciamento das identidades dos participantes e a proteção dos recursos contra aplicações. O gerenciamento de identidades é realizado usando os mesmos protocolos utilizados pelo projeto Globus [Foster and Kesselman 1998]. Atualmente a segurança dos recursos é feita limitando o acesso remoto a uma conta de login restrita (uma modificação da conta padrão *Unix*, *nobody*). Em versões anteriores, as soluções de segurança do Condor incluíram o uso de *chroot*, uma técnica de isolamento que restringe a execução de processos em porções seguras do sistema de arquivos. Esta técnica é considerada ineficaz, pois existem vulnerabilidades conhecidas que permitem aos processos isolados escaparem da barreira de segurança. Também já foi utilizada a instrumentação dinâmica do código em execução. Versões futuras do Condor poderão incorporar o uso de máquinas virtuais [VMCondor 2007].

4. Portifolio de Segurança

Soluções de segurança baseadas somente na confiança mútua não são viáveis para o OurGrid devido a sua natureza de livre entrada. Os usuários e recursos que fazem parte da grade não são signatários de qualquer acordo que possibilite a sanção de usuários que executarem ações maliciosas. Além disto, o sistema tem que suprir as necessidades de segurança de diversos perfis de usuários. Motivos como estes fazem com que um portifolio de segurança seja necessário.

Existem duas classes de problemas de segurança em grades: (i) a segurança dos usuários da grade contra recursos maliciosos; (ii) e a segurança dos recursos da grade

contra usuários maliciosos. Desta forma, cada componente do OurGrid apresenta desafios de segurança característicos de sua funcionalidade.

O Broker (usuário) deve ser responsável por validar e analisar os resultados obtidos pelos Workers (recursos), tentando mitigar qualquer sabotagem originada de um Worker malicioso. Neste caso, o OurGrid utiliza soluções de tolerância a sabotagem. O Worker, por sua vez, deve se proteger contra o uso indevido do recurso. No OurGrid, isto é obtido através do uso de soluções de *sandboxing*. Por terem funcionalidades distintas, as soluções de segurança no Broker são transparentes ao Worker e vice-versa, permitindo que cada componente adote soluções independentes.

Ademais, todos os componentes da grade precisam ter mecanismos para garantir que uma entidade não consiga se passar por outra. Isto é uma necessidade principalmente para os Peers, que são responsáveis pelo controle de recursos no sistema, nos quais uma identidade forjada pode burlar este controle. O OurGrid faz uso de técnicas de autenticação e autorização para garantir que os pares não tenham identidades forjadas.

No restante desta seção serão descritas as soluções de segurança que fazem parte do middleware OurGrid.

4.1. Worker: execução em máquinas virtuais

O conceito de virtualização, considerando o contexto deste trabalho, pode ser definido como a divisão dos recursos de um computador (hospedeiro) em diversos ambientes de execução virtuais (hóspedes ou máquinas virtuais). Soluções de virtualização são implantadas para melhor utilizar os recursos de servidores com múltiplos ambientes configurados para fins específicos, para execução de aplicações legadas, ambientes de teste de softwares, isolamento de recursos computacionais, etc [VMWare 2007]. Em um ambiente de grades computacionais, a virtualização é uma ferramenta de alta aplicabilidade. Devido a heterogeneidade dos recursos, o uso da virtualização pode fazer com que aplicações desenvolvidas para uma plataforma específica sejam executadas em outras dentro de uma máquina virtual. Também se pode fazer uso de virtualização para facilitar a gerência da grade e oferecer soluções de tolerância a faltas. Como argumentado na Seção 3, virtualização também pode ser utilizada para garantir o isolamento de tarefas em uma grade, aumentando a segurança do sistema. Existem diversos tipos de virtualização, cada um apresenta vantagens e desvantagens [Jones 2007]. Estes são descrito abaixo. Apresentamos na Figura 2 uma classificação dos tipos, considerando desempenho e nível de abstração.

Emulação Nesta abordagem a máquina hóspede executa dentro de um emulador de hardware. Com o uso de emulação cada instrução executada dentro do emulador é traduzida para uma ou mais instruções da máquina hospedeira. A maior vantagem da emulação é a capacidade de executar máquinas virtuais com arquiteturas de hardware diferentes da arquitetura da máquina hospedeira. Devido à necessidade de traduzir todas as instruções, a emulação tem um desempenho baixo em relação aos outros tipos.

Virtualização Total Virtualização total faz uso de um monitor de máquinas virtuais que pode executar instruções privilegiadas na máquina hospedeira. Uma máquina hóspede faz uso do monitor para executar as instruções feitas pela máquina hospedeira: caso sejam privilegiadas o monitor faz a devida tradução; caso não sejam,

o monitor executa a instrução direto no hardware. Embora tenha um desempenho melhor do que a emulação, esta abordagem é limitada já que necessita que as máquinas hóspede e hospedeira sejam da mesma arquitetura.

Paravirtualização Fazendo com que o sistema operacional executado na máquina hóspede seja consciente que está sendo executado em um ambiente virtualizado, é possível que o monitor de máquinas virtuais implemente um melhor mecanismo de tradução das chamadas feitas na máquina hóspede. Com isto, é possível alcançar um melhor desempenho ao custo do uso de sistemas operacionais modificados.

Virtualização no nível do Sistema Operacional Neste tipo de virtualização máquinas hóspedes do sistema usam o mesmo sistema operacional da hospedeira. Neste caso, o sistema operacional é capaz de limitar o acesso aos recursos feitos pela máquina hóspede, criando compartimentos isolados de execução. A maior vantagem deste tipo de virtualização é a pequena carga extra, pois não existe indireção com relação às instruções executadas na máquina hóspede.

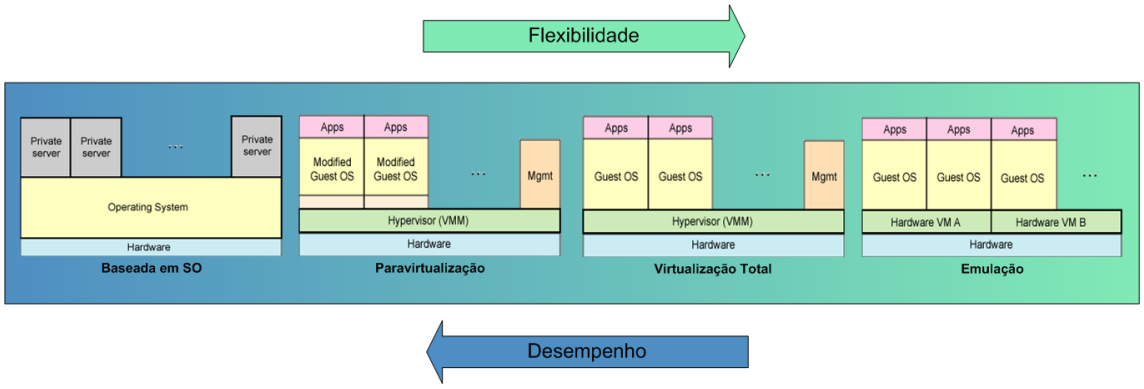

Figura 2. Diferentes tipos de Virtualização

No OurGrid optamos por criar ambientes seguros de execução de tarefas fazendo uso de virtualização, em especial utilizamos um recipiente de tarefas sem acesso à rede [Santhanam et al. 2005][1]. A Figura 3 esquematiza a seqüência de passos da execução de uma tarefa no compartimento seguro de execução. Antes do início da tarefa, toda entrada necessária é copiada do Broker para uma pasta compartilhada, que é acessível pela máquina virtual. Durante a execução da tarefa, a aplicação não tem maneira de se comunicar com os processos da máquina hospedeira nem tem acesso à rede, garantindo assim o seu isolamento. Após o término da aplicação, os dados de saída são copiados da pasta compartilhada para o Broker.

Para suprir a demanda de diferentes usuários, o OurGrid provê compartimentos seguros de execução baseados em diversas tecnologias de virtualização: QEMU[2] para oferecer emulação; VirtualBox[3] para virtualização total em computadores com arquitetura x86; XEN[4] que oferece paravirtualização também na arquitetura x86; e por fim, VServer[5]

[1] Os autores denominam este tipo de sandboxing de *Eager prefetching, whole-file-caching sandboxes*
[2] http://www.fabrice.bellard.free.fr/qemu/
[3] http://www.virtualbox.org/
[4] http://www.xensource.com/
[5] http://www.linux-vserver.org/

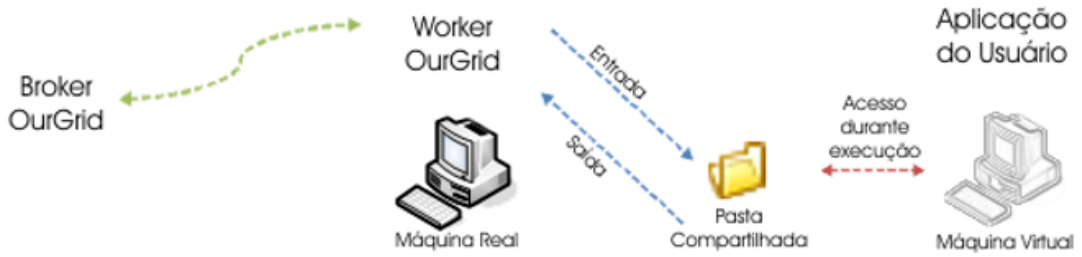

Figura 3. Esquema de funcionamento dos Compartimentos Seguros de Execução

para virtualização no nível do sistema operacional para Linux. O administrador de um domínio, no processo de implantação de um *Peer*, precisa tão somente escolher a versão do *Worker* diponibilizada com a tecnologia que achar mais adequada às restrições do seu domínio.

4.2. Broker: tolerância a sabotagem

Para proteger as aplicações dos usuários da grade contra recursos maliciosos, o OurGrid faz uso de uma solução de tolerância a sabotagem com uso de marcas d'água (também conhecido como códigos de certificação) similar à solução descrita em [Collberg and Thomborson 2002].

De forma resumida, o usuário da grade modifica sua aplicação adicionando uma marca d'água, esta introduz uma forma de identificar se o resultado da aplicação foi alterado. Um exemplo simples deste método é uma aplicação que computa uma série funções de saída não conhecida, dentre estas funções o usuário pode adicionar uma nova função de saída conhecida. Após o término da execução a saída da função conhecida é verificada para determinar se houve uma sabotagem. No OurGrid, o usuário adiciona à descrição de sua tarefa uma etapa adicional que contém o procedimento para verificação da marca d'agua, esta verificação é executada pelo Broker, o componente responsável pelo escalonamento de tarefas, que pode banir, por exemplo, um recurso tido como malicioso das próximas execuções.

Ademais, o portifolio de segurança também inclui a solução já existente de tolerância a sabotagem independente de aplicação [Oliveira and Brasileiro 2006], a qual exige um alto custo computacional, pois exige que para cada tarefa, réplicas sejam criadas e escalonadas em recursos confiáveis, a fim de comparar os resultados das mesmas tarefas escalonadas em nós desconhecidos. Esta solução pode ser adotada, se os requisitos da aplicação que está sendo executada na grade justificarem o custo computacional.

4.3. Autenticação e Autorização

O OurGrid faz uso de um mecanismo de incentivo para a troca de recursos [Andrade et al. 2007]. Nesse mecanismo, cada Peer contabiliza a doação dos seus recursos aos outros pares e o seu consumo dos recursos dos outros pares. Portanto, é preciso que cada um dos Peers seja identificado unicamente no sistema. Se for utilizado

um identificador público na rede do OurGrid, uma entidade maliciosa pode facilmente se passar por outro Peer. É preciso garantir que cada interação entre-pares tenha sua origem confirmada e associada ao identificador único de cada Peer, impedindo que mensagens forjadas, alteradas ou repedidas possam ser aceitas como válidas por qualquer componente do OurGrid.

Existem diversos mecanismos de autenticação que buscam contemplar tais funcionalidades, entretanto, alguns requerem o uso de servidores, como o Kerberos [Kohl and Neuman 1993], ou mesmo da configuração e uso de uma entidade certificadora para controlar a troca de identificações no sistema [Adams and Farrell 1999]. No OurGrid, usamos uma solução mais simples, próxima ao SPKI [Ellison et al. 1999], onde uma chave pública (de um par de chaves assimétricas [Kaliski 1998]) é a própria identidade de cada participante. Isto permite que o OurGrid mantenha suas características de livre entrada e fácil implantação. Note que cada Peer é autônomo para gerar sua identidade e o importante é que esta seja a mesma em todas as interações entre-pares do sistema. Uma característica inerente ao OurGrid é de que o mecanismo de incentivo garante que a troca de identidade de um Peer não é vantajosa, de modo que os Peers têm interesse em manter sua identidade.

Este mecanismo de segurança é implementado na camada de comunicação do OurGrid, o JIC. Cada mensagem trocada no sistema é assinada digitalmente através da chave privada e recebe em anexo a chave pública em questão, que serve para identificar o emissor da mensagem. O receptor por sua vez verifica a assinatura e, caso seja válida, pode então usar a chave pública para identificar a entidade emissora. O JIC usa chaves RSA [Jonsson and Kaliski 2003] de 1024 bits, pois é garantida, atualmente, a insuficiência de poder computacional para quebrar tal tipo de chave. Isto garante integridade e autenticação de cada mensagem. Usuários que desejem um maior nível de segurança podem fazer uso de entidades certificadoras para associar as identidades do sistema às identidades dos usuários, mas isso é uma funcionalidade opcional, já que dificulta a implantação do software.

Outras melhorias de segurança surgem ao se expandir o uso do identificador aos demais componentes do sistema. Por exemplo, pode-se garantir que um Worker só vai aceitar requisições do Broker ao qual foi alocado. Neste caso, ao alocar o Worker, o Peer envia uma mensagem para o mesmo com a identificação (chave pública) do Broker que vai utilizar os seus recursos. O Worker por sua vez só aceita pedidos de mensagens assinadas daquele Broker.

4.4. Assegurando o Código do Middleware

É necessário assegurar o código do middleware OurGrid contra ataques, como também bibliotecas e *containers* de terceiros utilizados na sua construção. Ataques ao código da aplicação, como *Buffer-Overflow*, são comuns e podem eventualmente prover acesso privilegiado para um atacante. Por outro lado, já descrevemos como o uso de virtualização limita as aplicações executadas na grade, agora também levantamos a necessidade de limitar os recursos utilizados pelos componentes do OurGrid para compor a grade. Caso um atacante consiga controlar de alguma maneira um dos componentes do OurGrid, este atacante pode fazer uso dos recursos que são providos para o componente.

O código do OurGrid é desenvolvido na linguagem Java, que é interpretada por

uma Máquina Virtual Java (*Java Virtual Machine ou JVM*). A linguagem Java e a JVM oferecem segurança no nível do código da aplicação [Sun Microsystems 2005]. Java também provê um conjunto de bibliotecas de segurança que permitem ao programador implementar diversos mecanismos para o desenvolvimento de aplicações seguras. Estas bibliotecas também são utilizadas pelo OurGrid como, por exemplo, os provedores de segurança.

Informações e detalhes sobre como a linguagem Java e seu interpretador provêem segurança podem ser encontradas na Internet no sítio sobre segurança em Java [Sun Microsystems 2007], em *white-papers* [Sun Microsystems 2005], como também na especificação da linguagem [Gosling et al. 2005] e da JVM [Lindholm and Yellin 1999].

5. Conclusão

Neste trabalho apresentamos um portifolio de segurança para um sistema de grade entre pares de livre entrada. Para definir o portifolio, fizemos um levantamento sobre grades computacionais, quais mecanismos de segurança são utilizados e são aplicados. Com este estudo foi possível levantar as necessidades de segurança do OurGrid e definir quais mecanismos de segurança devem fazer parte da sua solução. De fato, não existe uma solução única que atenda a todos os usuários de grades entre pares de livre entrada, portanto deve ser provido um portifolio com várias soluções de segurança.

Embora o portifolio implementado no OurGrid possa ser considerado abrangente, ainda pode ser expandido. Por exemplo, incorporar virtualização no nível do *hardware* [Habib 2008]. Esta tecnologia, a primeira das soluções de virtualização incorporadas no kernel estável do linux, implementa virtualização total se beneficiando da aceleração causada pelo suporte do processador.

Durante o desenvolvimento das soluções descritas nos deparamos com uma dificuldade adicional, a gerência de máquinas virtuais. A implantação e manutenção das imagens apresenta um alto custo administrativo. Uma possível solução para este problema seria a implantação de *VirtualWorkspaces* [Keahey et al. 2005] no OurGrid. Este serviço, além de facilitar a gerência das máquinas virtuais, expõe um protocolo de configuração que poderia ser usado pelo usuário da grade na criação de ambientes específicos para sua tarefa, por exemplo, com pacotes e utilitários que atendam de forma mais satisfatória suas submissões.

Agradecimentos

Este trabalho foi desenvolvido em colaboração com a HP Brasil P & D. Francisco Brasileiro é pesquisador do CNPq-Brasil.

Referências

Adams, C. and Farrell, S. (1999). RFC 2510: Internet X.509 public key infrastructure certificate management protocols. Request for Comments (RFC) 2510, Network Working Group.

Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. (2007). Automatic grid assembly by promoting collaboration in peer-to-peer grids. *Journal of Parallel and Distributed Computing*, 67(8):957–966.

Boinc Project (2007). Boinc project website, http://boinc.berkeley.edu/.

Boinc Status (2007). Boinc status, http://boincstats.com/stats/project_graph.php?pr=sah.

Cappello, F., Djilali, S., Fedak, G., Hérault, T., Magniette, F., Néri, V., and Lodygensky, O. (2005). Computing on large-scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. *Future Generation Comp. Syst*, 21(3):417–437.

Cavalcanti, E., Assis, L., Gaudencio, M., Cirne, W., Brasileiro, F., and Novaes, R. (2006). Sandboxing for a free-to-join grid with support for secure site-wide storage area. In *1st International Workshop on Virtualization Technology in Distributed Computing*.

Collberg, C. S. and Thomborson, C. (2002). Watermarking, tamper-proofing, and obfuscation - tools for software protection. *Software Engineering, IEEE Transactions on*, 28(8):735–746.

Distributed.net (2007). Distributed.net project, http://www.distributed.net/.

Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., and Ylonen., T. (1999). RFC 2693: Spki certificate theory. Request for Comments (RFC) 2693, Network Working Group.

Engine, S. G. (2007). Sun grid engine project web page, http://gridengine.sunsource.net/.

Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S. (1998). A security architecture for computational grids. In *ACM Conference on Computers and Security*, pages 83–91. ACM Press.

Foster, I. T. and Kesselman, C. (1998). The globus project: A status report. In *Heterogeneous Computing Workshop*, pages 4–18.

Frey, J., Tannenbaum, T., Livny, M., Foster, I. T., and Tuecke, S. (2002). Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246.

gLite (2007). glite project web page, http://glite.web.cern.ch/glite/.

Gosling, J., Joy, B., Steele, G., and Bracha, G. (2005). *Java(TM) Language Specification, The (3rd Edition) (Java Series)*. Addison-Wesley Professional.

Habib, I. (2008). Virtualization with kvm. *Linux J.*, 2008(166):8.

Jones, M. T. (2007). Virtual linux, http://www.ibm.com/developerworks/library/l-linuxvirt/index.html.

Jonsson, J. and Kaliski, B. (2003). RFC 3447: Public-key cryptography standards (pkcs) #1: Rsa cryptography specifications version 2.1. Request for Comments (RFC) 3447, Network Working Group.

Kaliski, B. (1998). RFC 2315: Pkcs #7: Cryptographic message syntax. Request for Comments (RFC) 2315, Network Working Group.

Kamp, P. H. and Watson, R. N. M. (2000). Jails: Confining the omnipotent root. In *In Proceedings of the 2nd International SANE Conference*.

Kapadia, N. H., Figueiredo, R. J. O., and Fortes, J. A. B. (2000). PUNCH: Web portal for running tools. *IEEE Micro*, 20(3):38–47.

Keahey, K., Foster, I., Freeman, T., Zhang, X., and Galron, D. (2005). Virtual Workspaces in the Grid. *Lecture Notes in Computer Science*, 3648:421–431.

Kohl, J. T. and Neuman, B. C. (1993). The Kerberos network authentication service (V5), citeseer.ist.psu.edu/article/kohl93kerberos.html. Technical Report 1510.

Lima, A., Cirne, W., Brasileiro, F., and Fireman, D. (2006). A case for event-driven distributed objects. In *8th International Symposium on Distributed Objects and Applications (DOA)*, pages 1705–1721, Berlin / Heidelberg. Springer.

Lindholm, T. and Yellin, F. (1999). *The Java(TM) Virtual Machine Specification (2nd Edition)*. Prentice Hall PTR.

Litzkow, M. J., Livny, M., and Mutka, M. W. (1988). Condor - A hunter of idle workstations. In *ICDCS*, pages 104–111.

Neary, M. O., Christiansen, B. O., Cappello, P. R., and Schauser, K. E. (1999). Javelin: Parallel computing on the internet. *Future Generation Comp. Syst*, 15(5-6):659–674.

Oliveira, A. C. and Brasileiro, F. (2006). Escalonamento tolerante a sabotagem para grades computacionais entre-pares. In *VII Workshop de Testes e Tolerância a Falhas (WTF) in conjunction with Simpósio Brasileiro de Redes de Computadores (SBRC)*.

Ong, T. M., Lim, T. M., Lee, B. S., and Yeo, C. K. (2002). Unicorn: voluntary computing over Internet. *Operating Systems Review*, 36(2):36–51.

OpenPBS (2007). Open source workload management software, http://www.openpbs.org/.

Santhanam, S., Elango, P., Arpaci-Dusseau, A., and Livny, M. (2005). Deploying virtual machines as sandboxes for the grid. In *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, Berkeley, CA, USA. USENIX Association.

Sarmenta, L. F. G. (1998). Bayanihan: Web-based volunteer computing using Java. *Lecture Notes in Computer Science*, 1368.

Sarmenta, L. F. G. (2001). Sabotage-tolerance mechanisms for volunteer computing systems. In *CCGRID*, pages 337–346. IEEE Computer Society.

Sun Microsystems (2005). Java security overview - white paper, http://java.sun.com/developer/technicalarticles/security/whitepaper/js_white_paper.pdf.

Sun Microsystems (2007). Java se security website, http://java.sun.com/developer/technicalarticles/security/whitepaper/js_white_paper.pdf.

TeraGrid (2007). Teragrid project web page, http://www.teragrid.org/.

Thain, D., Tannenbaum, T., and Livny, M. (2005). Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356.

TOP 500 (2007). Top 500 supercomputer sites, http://www.top500.org/.

VMCondor (2007). Virtual machines in condor, http://www.cs.wisc.edu/condor/pcw2007/condor_presentations.html/.

VMWare (2007). Virtualization overview, http://www.vmware.com/files/elqnow/elqredir.htm?ref=http://www.vmware.com/pdf/virtualization.pdf.