A Usage Control Platform based on Rule Templates and Authorization Credentials

Maicon Stihler, Altair O. Santin, Arlindo L. Marcon Jr.
Programa de Pós-Graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba – PR – Brasil

Abstract

The popularization of PaaS environments presents challenges to traditional authorization models for controlling resource usage. This article describes a hybrid authorization model suitable for PaaS environments, using rule templates, authorization credentials and local derivation of individual policies. In addition, we present the usage control mechanisms that eliminate the main disadvantages of policy models based on provisioning and outsourcing. The work also shows the technical details of the implementation of a prototype with its performance evaluation results.

Keywords

Controle de Uso; UCON$_{ABC}$; Gabaritos de Regras; Credenciais de Autorização;

I. Introdução

O termo Computação em Nuvem [1] se refere à utilização de diversas tecnologias para criar um ambiente dinâmico, onde organizações podem implementar seus próprios serviços usando infraestrutura de TI sob demanda. Isto é, as organizações podem criar serviços que se adaptam dinamicamente às suas necessidades, através do emprego de uma infraestrutura elástica (redimensionável). Tais recursos podem ser utilizados sem a necessidade de instalação e configuração de equipamentos e softwares especializados, sendo acessíveis através de qualquer computador conectado à Internet.

Para que isso seja possível, a computação em nuvem está organizada em três modelos de serviço bem definidos, de acordo com o nível de abstração da funcionalidade oferecida [2]. São eles: Infraestrutura como serviço (IaaS), Plataforma como Serviço (PaaS) e Software como Serviço (SaaS). O modelo de IaaS oferece recursos computacionais básicos (e.g. armazenamento, processamento e redes) como serviço, permitindo que as organizações instalem e executem softwares à sua escolha sobre uma infraestrutura virtual. Os ambientes de PaaS, oferecem serviços intermediários (e.g. segurança, gerenciamento de usuários) que podem ser utilizados para o desenvolvimento e execução de aplicações, ocultando a infraestrutura subjacente. No modelo de SaaS, os usuários utilizam aplicações prontas disponibilizadas pelo provedor. Esses modelos são conceitualmente independentes entre si, sendo possível, mas não mandatório, utilizar os serviços de um modelo (e.g. IaaS) para criar um ambiente de maior abstração (e.g. PaaS ou SaaS).

De acordo com [3], por atender um público heterogêneo, com serviços diversos e com requerimentos diferentes, a computação em nuvem exige políticas de controle de acesso que sejam de granularidade fina. Os mecanismos de controle de acesso devem ser capazes de capturar a dinamicidade da nuvem, utilizando informações contextuais, de atributos e credenciais. Adicionalmente, é desejável que a nuvem ofereça controles que permitam capturar aspectos relevantes de nível de serviço (SLA), enquanto oferecem meios de gerenciamento amigáveis e eficientes.

Para atender à essa demanda por dinamismo, uma das tendências tem sido a utilização do modelo de controle de uso, UCON$_{ABC}$ [4], por sua capacidade de refletir mudanças (e.g. em atributos de usuários, objetos, ambiente) em tempo de execução. Pesquisas anteriores para aplicação de UCON$_{ABC}$ em nuvens computacionais ([5], [6], [7]), têm utilizado abordagens centralizadoras, isto é, baseadas no modelo de *outsourcing* e, por isso, herdam suas principais desvantagens: os custos de comunicação de rede (*overhead*), ponto único de falha no monitor de referência e baixa escalabilidade. Em [5] é proposta a utilização de *espaços de tuplas* para minimizar esses problemas, porém, tal abordagem não é baseada em padrões abertos e requer a implementação de mecanismos complexos, como demonstrado por Capizzi e Messina [8], [9]. Abordagens descentralizadas baseadas em *provisioning* de políticas, porém, também apresentam dificuldades relacionadas à sincronização das políticas no ambiente distribuído.

Este artigo descreve um sistema para gerência de controle de uso no modelo PaaS. Para isso, utiliza uma abordagem híbrida de *provisioning* com gabaritos de regras previamente configurados nos servidores e credenciais de autorização gerenciadas separadamente. As políticas individuais são derivadas localmente, isto é, no servidor onde os mecanismos de avaliação e aplicação se encontram. Isto ocorre através da combinação de informações contidas nas credenciais com os gabaritos de regras instalados no servidor. A proposta elimina a necessidade de sincronização de políticas locais com políticas remotas, presente nos modelos baseados em *provisioning* tradicional, possibilitando a geração de políticas individuais diferenciadas e a melhora da frequência de reavaliação das regras de controle de uso em comparação com as abordagens centralizadas (*outsourcing*).

978-1-4673-7767-6/15 $31.00 © 2015 IEEE
DOI 10.1109/SBRC.2015.17

O artigo está organizado da seguinte maneira: Seção II apresenta a fundamentação; A Seção III discute a proposta em detalhes; A implementação de um protótipo e testes de avaliação são descritos na Seção IV; Os trabalhos relacionados estão na Seção V e a Seção VI apresenta as conclusões deste trabalho.

II. Fundamentação

Esta seção apresenta uma breve discussão sobre conceitos fundamentais para a compreensão deste trabalho. Nomeadamente, serão abordados os conceitos de plataforma como serviço, arquiteturas de políticas e o modelo de controle de uso.

A. Ambientes de Plataforma como Serviço

O ambiente de nuvem pode oferecer modelos de serviço com níveis de abstração que vão do mais baixo (infraestrutura) até o mais alto (software), sendo o modelo intermediário a Plataforma como Serviço (PaaS). De acordo com [2], um ambiente de PaaS se caracteriza por abstrair algumas funcionalidades uteis de uso comum para o usuário. Ele pode ser implementado como um ambiente de execução, com bibliotecas que podem ser empregadas pelo usuário para desenvolver aplicações que consomem serviços oferecidos pela plataforma. Os recursos subjacentes são geridos de maneira transparente pela plataforma, isto é, o usuário não tem como gerenciar tais recursos diretamente. É possível, no entanto, controlar a execução de sua aplicação.

Um aspecto característico de ambientes PaaS é a utilização de mecanismos mais leves para isolamento de aplicações. Isto pode ser observado em soluções populares como o Red Hat OpenShift [10], Heroku Cloud Platform [11] e Cloud Foundry [12]. Um desses mecanismos é conhecido como virtualização em nível de sistema operacional ou baseada em *containers* [13]. Trata-se de uma forma menos abrangente de virtualização, para situações em que o custo de virtualização em hardware, como geralmente utilizado no modelo IaaS, é muito alto. Pode ser utilizado em ambientes de computação de alto desempenho, *clusters*, *grid computing*, *web hosting*, etc.

Este tipo de isolamento oferece maior desempenho bruto e maior escalabilidade. Algumas avaliações experimentais mostraram que em algumas situações a sobrecarga de virtualização completa é muito maior do que a baseada em *containers* [14], [15]. Por essa razão, é uma opção popular para a construção de ambientes de PaaS, permitindo a execução de aplicações concorrentes em um mesmo hospedeiro, sem que as aplicações tenham conhecimento umas das outras.

B. Arquiteturas de Políticas

As arquiteturas para configuração e avaliação de políticas têm sido tradicionalmente implementadas de acordo com duas abordagens diferentes, que podem ser resumidas como *provisioning* e *outsourcing*:

Provisioning exige configuração das políticas nos mecanismos de controle local. A política fica em um repositório local. O guardião do recurso consulta um ponto de decisão de política (monitor de referência) local para obter uma decisão e fazer sua aplicação (*enforcement*) [16]. Sua vantagem é a robustez, pois não tem dependência externa, porém sua desvantagem é a complexidade para manter atualizadas as políticas distribuídas no ambiente.

Outsourcing atua como um modelo cliente e servidor. Um serviço externo, o monitor de referência, responde a pedidos de avaliação de políticas de todos os guardiões de recursos do ambiente. Cada tentativa de acesso exige uma avaliação de política no serviço remoto. O guardião somente aplica a decisão [17]. Sua vantagem é simplificar a atualização das políticas e a implementação do guardião. Suas desvantagens são o custo extra de comunicação e sua fragilidade, pois o avaliador de políticas é um ponto único de falha. Além disso, por ser centralizado, possui maior complexidade inerente para obter escalabilidade compatível com ambientes de nuvem computacional.

C. Controle de Uso

O modelo UCON$_{ABC}$ [4] é a reunião de diversas ideias anteriores para o controle de uso de recursos em um modelo formal e coeso. Ele permite a representação de controles como *Digital Rights Management* (DRM), discricionários, mandatórios, baseados em papéis e baseados em confiança (*trust management*).

UCON$_{ABC}$ possui dois conceitos fundamentais: (i) Continuidade, define que os controles definidos na política sejam avaliados e aplicados durante todo o uso do recurso (antes de iniciar/*pre*, durante/*ongoing* e ao fim/*post*); (ii) Mutabilidade, significa que determinados atributos do sujeito ou objeto podem ser alterados pela política de controle durante o uso.

Os controles são categorizados como sendo autorizações (o usuário precisa ter os direitos para a execução de uma ação), condições (avaliações feitas sobre atributos do ambiente, como a hora ou localização geográfica), obrigações (ações externas, que não estão nem em nível de autorização e nem de condições) e atualizações (modificações que se refletem nos atributos de usuário ou objeto).

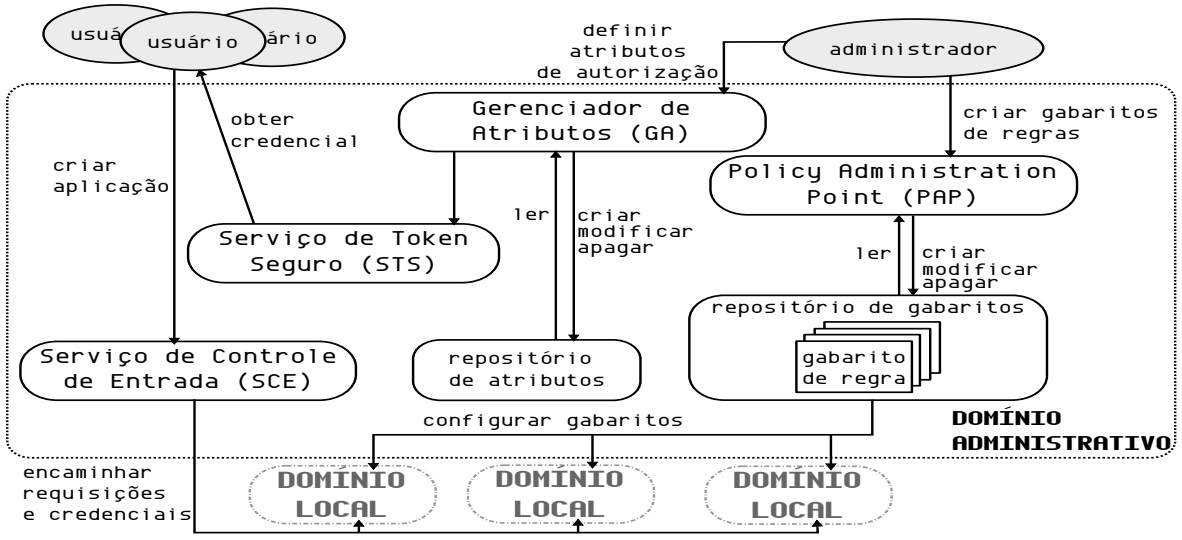

Figura 1. Componentes do Domínio Administrativo

III. Gerência Distribuída de Controle de Uso em Plataformas como Serviço

Nesta seção é proposta uma infraestrutura para gerenciamento de controle de uso para PaaS, que utiliza gabaritos de regras e credenciais de autorização para derivar políticas que são avaliadas localmente. A abordagem proposta elimina a necessidade de configuração direta de políticas existente em arquiteturas baseadas em *provisioning*, assim como o custo de comunicação presente em arquiteturas baseadas em *outsourcing*.

Os componentes que compõe a infraestrutura podem ser categorizados como sendo de *domínio administrativo* ou de *domínio local*. No nível administrativo estão os serviços que possibilitam a gerência dos atributos de autorização que compõe a credencial de autorização, e a criação dos gabaritos de regras que são previamente configurados nos *domínios locais* (*hosts*) onde as aplicações são executadas. No nível local estão os componentes para validação de credenciais de autorização, derivação da política de controle de uso, avaliação de políticas, e aplicação da decisão gerada no monitor de referência.

As aplicações de usuário são executadas localmente em ambientes isolados com mecanismos do sistema operacional, que permitem contabilização e controle finos, independente do número de processos em execução dentro deste ambiente isolado. Cada ambiente isolado é subordinado a uma política individual de controle de uso, derivada a partir da credencial de autorização apresentada e dos gabaritos de regras preexistentes no ambiente.

A infraestrutura foi pensada para eliminar a necessidade de configuração de políticas para cada aplicação, assim como a consequente necessidade de rastrear essas políticas para que possam ser sincronizadas quando alguma modificação se faz necessária. Os mecanismos de controle de uso locais permitem um tempo de resposta mais curto que em abordagens baseadas em *outsourcing*, além de eliminar o ponto único de falha próprio destas abordagens: falhas ou mudanças em algum *domínio local* não apresentam impactos no resto do ambiente.

A. Domínio Administrativo

O domínio administrativo é constituído de quatro serviços: o *Policy Administration Point* (PAP, ou ponto de administração de política), onde são criados os gabaritos de regras; o gerenciador de atributos (GA), utilizado para definir os atributos de autorização aplicáveis a cada usuário; o *Security Token Service* (STS, ou serviço de *token* de segurança), que emite credenciais de autorização; o serviço de controle de entrada (SCE), que previne que uma credencial de autorização seja indevidamente utilizada em múltiplos *domínios locais*. Esses componentes estão ilustrados na Figura 1.

O PAP tem a função de gerenciar um repositório de gabaritos de regras. Cada gabarito refere-se a uma única regra de controle de uso, tendo lacunas que são identificadas por variáveis de nome único (o *id-lacuna*). Esses gabaritos devem representar todas as regras que o administrador deseja impor no ambiente protegido como, por exemplo, regras para controle de horário, utilização de disco, número de instâncias, etc. Além disso, a utilização de gabaritos de regras, ao invés de gabaritos de políticas, se dá pela necessidade de manter a flexibilidade: usuários diferentes podem ser submetidos a combinações diferentes destes gabaritos.

O repositório de gabaritos de regras é configurado em todos os domínios locais – um exemplo concreto consiste em copiar tal repositório na imagem do sistema operacional das máquinas virtuais onde as aplicações serão executadas, de modo que o repositório esteja disponível imediatamente após a inicialização do sistema. O fato dos gabaritos representarem padrões de regras comuns ao ambiente, permite deduzir que atualizações neste repositório serão raras. No entanto, os repositórios contidos nos domínios locais podem ser atualizados a qualquer momento, mediante um

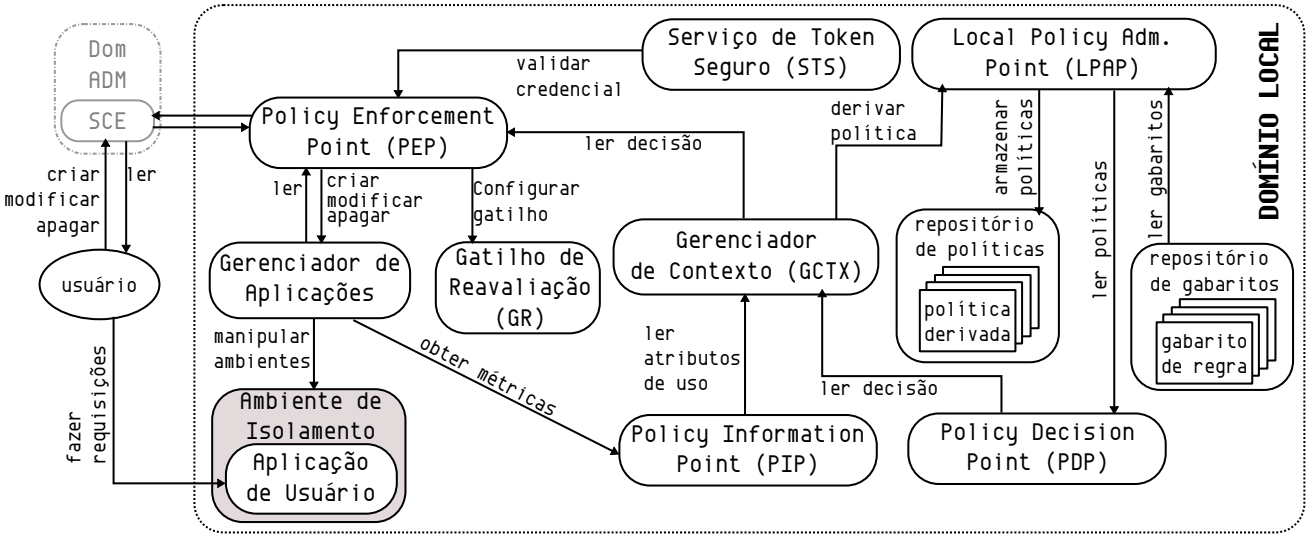

Figura 2. Componentes do Domínio Local

sistema de notificação (e.g. quando um gabarito muda no domínio administrativo, ele pode ser difundido para todos os participantes através de uma mensagem de *broadcast* confiável. Os domínios locais atualizam seus repositórios e re-derivam as políticas).

O serviço de GA consiste em um repositório de informações que irão compor a credencial de autorização. Entre as informações gerenciadas está um conjunto de identificadores (chamados *id-lacuna*), que são organizados em subgrupos correspondentes aos tipos de controles UCON$_{ABC}$ (*pre* e *ongoing*). Esse conjunto irá definir quais gabaritos de regra serão selecionados para derivar a política de controle de uso a ser aplicada. Além disso, o repositório mantém registros das quantidades de recursos que podem ser concedidas a cada usuário, além do montante de recursos que foram concedidos a cada usuário nas credenciais já emitidas. Desta maneira, um usuário pode estar habilitado para uma determinada quantia de recurso (e.g. 100 GB de disco) e ter solicitado apenas uma parcela (e.g. 10 GB), o que permite que esse usuário solicite outras credenciais futuramente. É deste modo que o GA impede que um usuário solicite mais recursos do que aqueles que lhe foram alocados pelo administrador. O conjunto de *id-lacuna*s e os atributos de autorização, formam a credencial que, após ser combinada com os gabaritos de regra da origem a política de controle de uso aplicável a requisição do usuário.

O STS, por sua vez, é o serviço que confecciona a credencial de autorização a pedido do usuário. Para que isso ocorra, o usuário deve enviar um pedido de emissão de credencial ao STS. Após a autenticação do pedido, o STS contacta o GA e solicita a validação das quantidades pedidas pelo usuário: uma solicitação inválida (e.g. por exceder a quantidade permitida) é prontamente recusada. O GA, ao identificar um pedido válido, registra a quantidade solicitada no montante de recursos concedidos ao usuário e retorna uma confirmação ao STS, contendo os atributos solicitados e o conjunto de identificadores de gabaritos de regras aplicável ao usuário. O STS constrói uma credencial contendo estas informações, cifrando a credencial com a chave pública do SCE, de modo que o usuário não saiba quais os detalhes contidos no documento. A credencial gerada é enviada ao usuário que, a partir deste momento, pode invocar a infraestrutura para solicitar seus serviços.

O último serviço do domínio administrativo é o SCE, ou serviço de controle de entrada. Esse componente funciona como o portão de entrada da plataforma, que pode contar com inúmeros *hosts* para a execução de aplicações de usuário, tornando inviável a emissão de credenciais específicas para apenas um *domínio local* (e.g. caso o *host* escolhido esteja com problemas, o usuário só descobrirá isso ao tentar realizar uma requisição, tendo que solicitar a invalidação da credencial atual e a emissão de outra credencial para outro *host*). A emissão de credenciais genéricas de autorização exige, entretanto, que se controle o uso da credencial para que não seja apresentada em dois ou mais lugares simultaneamente. Por estas razões, o SCE tem duas funções: selecionar um *domínio local* em boas condições para atender a requisição do usuário e rastrear onde as credenciais estão sendo usadas, de modo a impedir que tentativas de uso em múltiplos guardiões ocorram.

Os componentes do domínio administrativo possuem menor criticidade em relação ao funcionamento dos domínios locais. Isto é, pequenos momentos de indisponibilidade não irão afetar o funcionamento das aplicações que já estão em execução, pois todos os mecanismos de controle de uso estão no *domínio local*. Contudo, é interessante que os serviços deste nível adotem estratégias para alta disponibilidade, de forma que possíveis problemas neste nível não impeçam que os usuários iniciem novas aplicações ou modifiquem os parâmetros daquelas que já se encontram em execução.

Procedimento 1 Derivação de Política

1: $PS \leftarrow grupos\_de\_politicas(credencial)$
2: $DS \leftarrow cria\_novo\_agrupamento()$
3: **enquanto** $PS \neq \emptyset$ **faça**
4: $P \leftarrow retira\_proxima\_politica(PS)$
5: $D \leftarrow cria\_nova\_politica(P)$
6: $IS \leftarrow ids\_de\_gabarito(P)$
7: **enquanto** $IS \neq \emptyset$ **faça**
8: $ID \leftarrow retira\_proximo\_id(IS)$
9: $GI \leftarrow busca\_gabarito(ID, repositorio)$
10: **se** $GI = \emptyset$ **então**
11: **retorna** falha
12: **fim se**
13: $configura(GI, credencial)$
14: $adiciona(GI, D)$
15: **fim enquanto**
16: $adiciona(D, DS)$
17: **fim enquanto**
18: $salva(DS, repositorio)$
19: **retorna** sucesso

B. Domínio Local

O domínio local é composto por todos os mecanismos de controle de uso presentes no sistema local. É nele que a aplicação de usuário é executada em um ambiente isolado, onde as credenciais de autorização são validadas e combinadas com gabaritos de regras, para gerar a política de controle de uso. A avaliação e aplicação desta política é realizada de modo completamente independente de mecanismos externos. A Figura 2 ilustra os componentes que compõe o mecanismo local. Cada domínio local, presente na Figura 1, consiste de um servidor capaz de executar aplicações de usuário e realizar o controle de uso das mesmas, como será mostrado a seguir.

O usuário pode realizar duas modalidades de requisição: de aplicações e de gerenciamento. O primeiro tipo são requisições dependentes do tipo de aplicação que o usuário está executando (e.g. requisições GET para um servidor web). As requisições de gerenciamento são utilizadas para criar, modificar, apagar ou obter informações sobre uma determinada aplicação (e.g. solicitação para executar uma aplicação web). Os mecanismos de controle locais estão focados unicamente no segundo tipo de requisições.

O primeiro item a ser observado na Figura 2 é o serviço de controle de entrada (SCE). Quando o usuário deseja executar uma requisição de gerenciamento, como criar ou modificar uma aplicação, o pedido deve ser encaminhado ao SCE, juntamente com a credencial do usuário. O SCE atua como um despachante de pedidos, externo ao domínio local, que sabe para qual domínio deve encaminhar a requisição de usuário.

O guardião do recurso, conhecido como *Policy Enforcement Point* (PEP), recebe as requisições encaminhadas pelo SCE e deve garantir que estas sejam executadas somente se forem autorizadas. Para que isso seja possível, o primeiro passo a ser realizado consiste na validação da credencial do usuário.

A validação é realizada no serviço de *token* de segurança (STS). Nesse estágio são verificadas as datas de validade, autenticidade das assinaturas, relações de confiança, integridade dos dados, entre outros. Qualquer falha na validação da credencial faz com que a requisição do usuário seja considerada inválida.

Considerando que o PEP obtenha uma resposta positiva do STS, o próximo passo é submeter uma requisição de autorização ao gerenciador de contexto (GCTX) juntamente com informações contextuais (e.g. credencial do usuário, requisição solicitada, argumentos de requisição, etc). O PEP fica aguardando uma resposta do GCTX com a decisão de autorização a ser aplicada.

O gerenciador de contexto é o componente que atua como integrador das partes do domínio local. Sua primeira tarefa é interpretar o formato de comunicação utilizado pelo PEP, convertendo-o para um formato útil aos demais componentes. Após extrair a credencial de política da requisição enviada pelo PEP, o GCTX envia uma requisição para derivação de política ao gerenciador de políticas local, ou *local policy administration point* (LPAP). Em paralelo o GCTX recupera quaisquer informações de utilização de recursos associadas ao usuário, aplicação (caso já esteja em execução) e ambiente através do repositório de informações (*Policy Information Point*, PIP).

O LPAP tem a função de confeccionar políticas de controle de uso derivadas da credencial de usuário e das regras contidas no repositório de gabaritos. O Procedimento 1 resume as ações executadas pelo LPAP para derivar uma política. O LPAP identifica quais grupos de políticas devem ser criados (e.g. *pre-authorization*, *ongoing-conditions*, etc) e guarda

Procedimento 2 Avaliação de Política

1: $S \leftarrow sujeito(credencial)$
2: $O \leftarrow objeto(credencial)$
3: $C \leftarrow contexto(credencial)$
4: $E \leftarrow endereco(LPAP)$
5: $PS \leftarrow recupera\_politica(S, O, E)$
6: **se** $PS = \emptyset$ **então**
7: **retorna** falha
8: **fim se**
9: **enquanto** $PS \neq \emptyset$ **faça**
10: $P \leftarrow retira\_proxima\_politica(PS)$
11: **enquanto** $P \neq \emptyset$ **faça**
12: $R \leftarrow retira\_proximo\_regra(P)$
13: **se** $avalia(R, S, O, C) = Negado$ **então**
14: **retorna** falha
15: **fim se**
16: **fim enquanto**
17: **fim enquanto**
18: **retorna** sucesso e gatilho de reavaliação

em PS. As políticas que serão criadas são agrupadas em um único documento, DS. O LPAP cria um novo documento de política para cada grupo de PS e guarda em D. Os gabaritos que devem ser aplicados são descobertos através da inspeção da lista de identificadores contida em cada entrada P. O LPAP recupera, uma a um, os gabaritos correspondentes e executa um processo de configuração dos campos vazios, devidamente identificados, com as informações contidas na credencial do usuário. Após configurar todos os gabaritos, o documento da política é adicionado ao agrupamento DS (linha 16). Quando não houverem mais políticas em PS, a política resultante (contida em DS) é armazenada no repositório de políticas. Uma mensagem de sucesso é retornada ao GCTX, permitindo que o processo de avaliação de autorização continue.

O PIP, por outro lado, é o componente responsável por disponibilizar informações de uso ao GCTX, através de uma interface unificada. As informações de uso são coletadas por outros componentes como, por exemplo, o gerenciador de aplicações. Os agentes de coleta de métricas utilizam as interfaces nativas do sistema operacional para contabilizar o consumo realizado por aplicações individuais, assim como o estado atual do sistema (e.g. o *load average*, número de aplicações em execução, etc). Esses dados são essenciais para a correta avaliação das políticas de controle de uso.

Antes de solicitar a avaliação de política, GCTX inspeciona a requisição para verificar se ela se aplica a uma aplicação já em execução ou se trata-se de uma nova instância. Isso é essencial para que a fase correta do controle de uso seja aplicada. Como os ambientes de isolamento representam o objeto a ser controlado, sua existência implica uma sessão de uso já em execução. O GCTX utiliza essa informação para solicitar uma decisão ao mecanismo de decisão local (*Policy Decision Point*, PDP) e fica aguardando pela resposta.

O PDP é o componente especializado em interpretar as regras de políticas e verificar se as regras são atendidas com a credencial apresentada pelo usuário dentro das condições estabelecidas, ver Procedimento 2. Para que possa fazer isso, o PDP deve recuperar a política aplicável ao usuário e aplicação desejada. Essa política é obtida através do LPAP, que utiliza o identificador do usuário e aplicação para recuperar o documento apropriado no repositório de políticas. A ausência de políticas faz com que a requisição seja automaticamente negada (por este motivo o GCTX deve aguardar a confirmação LPAP após a derivação da política). De posse da política o PDP combina as informações contextuais enviadas pelo GCTX e verifica se a política está sendo respeitada. A decisão obtida é retornada ao GCTX juntamente com os detalhes de aplicação da mesma.

O GCTX converte a resposta obtida do PDP para um formato compreensível pelo PEP. Nessa resposta se encontram a decisão que deve ser aplicada (autorizado ou negado) e o gatilho de reavaliação da política, isto é, a frequência com que se deve reavaliar a política. O PEP configura o gatilho de reavaliação (GR) no componente que recebe o mesmo nome. O GR funciona como um alarme que informa ao PEP quando uma política deve ser reavaliada. Quando o gatilho é criado, o PEP repassa os dados da requisição sendo que, no momento da reavaliação, o PEP terá informações referentes ao reinício do processo de reavaliação. O PEP encaminha requisições autorizadas para o gerenciador de aplicações local, que fará a execução efetiva da requisição do usuário. A resposta obtida nesse estágio é retornada ao usuário (e.g. informações de acesso ip:porta).

Como o modelo proposto é independente da tecnologia escolhida, o gerenciador de aplicações implementa as ações necessárias para realizar o pedido do usuário. Resumidamente, é criado um novo ambiente isolado no sistema operacional,

```
1 <Rule RuleId="CPURule" Effect="Permit">
2   <Target><Any/></Target>
3   <Condition>
4     <Apply FunctionId="integer-less-than-or-equal">
5       <Apply FunctionId="integer-one-and-only">
6         <AttributeDesignator Category="access-subject"
7         AttributeId="usedCpu" DataType="integer"/>
8       </Apply>
9       <Apply FunctionId="integer-one-and-only">
10        <AttributeValue DataType="integer">
11          ${TotalCpuTime}
12        </AttributeValue>
13      </Apply>
14    </Apply>
15  </Condition>
16 </Rule>
```

Figura 3. Exemplo de Gabarito de Regra

o limite de CPU e disco é configurado de acordo com os parâmetros recebidos, assim como os detalhes de acesso a rede. O gerenciador instala a aplicação no ambiente isolado e a executa. Se não encontrar nenhum erro, os dados de acesso são retornados, caso contrário uma mensagem de erro é retornada.

C. Gabaritos, Derivação e Atualizações

Cada gabarito de regra representa um controle bem definido. Na Figura 3 apresentamos uma versão simplificada de um gabarito para controle de tempo de processamento (CPU). O primeiro item a ser observado é o identificador *RuleID*: este valor identifica o gabarito unicamente, sendo que a credencial do usuário carrega esse valor como um de seus atributos. O segundo aspecto importante é o identificador *TotalCpuTime*: é o parâmetro que deve ser configurado para cada usuário, isto é, o *id-lacuna* discutido anteriormente. Quando o usuário solicita a emissão de uma credencial, ela conterá um atributo com o identificador *TotalCpuTime* indicando o limite de uso de CPU. O terceiro aspecto é a variável *usedCpu*, este atributo não é modificado com dados da credencial, pois é obtido localmente através do PIP – é a informação de contabilidade local.

Cada credencial carrega os identificadores de gabaritos separados por grupos (e.g. grupos *pre* e *ongoing*). Os atributos de configuração são carregados em uma seção separada. Como exibido na Figura 3, a derivação da regra consiste em recuperar o gabarito correto (identificado pelo *RuleID*) e substituir os campos (identificados por ${}) com o valor dos atributos correspondentes das credenciais. As regras resultantes são organizadas em políticas separadas, de acordo com a fase de aplicação, e armazenadas em um documento agrupador (*PolicySet*). Para permitir a individualização, os *PolicySets* e suas políticas, possuem um alvo (*Target*) aplicável somente ao usuário e a aplicação da requisição (identificada por um ID único).

Ao salvar as políticas derivadas no repositório, o LPAP registra os *metadados* da política, que inclui, entre outras informações, os identificadores dos gabaritos que deram origem. Desta maneira, supondo que um ou mais gabaritos sejam modificados pelo administrador, o LPAP tem como identificar quais políticas dependiam daquele gabarito e desencadear a sua re-derivação a partir dos gabaritos atualizados.

IV. Implementação e Avaliação

O sistema proposto foi avaliado através da implementação de um protótipo dos componentes utilizados no domínio local. Isto permitiu verificar se os algoritmos locais são capazes de implementar os mecanismos de controle de uso desejados. Além disso, foi possível fazer uma análise de desempenho do avaliador de políticas (PDP) e identificar a diferença do tamanho das mensagens transferidas na proposta com mensagens que seriam utilizadas em abordagens baseada em *provisioning* puro e *outsourcing*.

A. Implementação do Protótipo

Para a implementação do protótipo foram utilizados alguns projetos de código livre, que foram integrados através da linguagem de programação Java. A compartimentalização de aplicações de usuários é realizada através das *jails* do sistema FreeBSD [18], que oferecem os mecanismos para monitorar o consumo de recursos da aplicação e para gerenciar a execução da mesma. As credenciais de autorização seguem a especificação SAML [19], usando mensagens do tipo *AttributeStatement* que são criadas e manipuladas através do projeto OpenSAML [20]. As políticas de controle de acesso seguem o padrão XACML [21] e são avaliadas através da biblioteca WSO2 Balana [22].

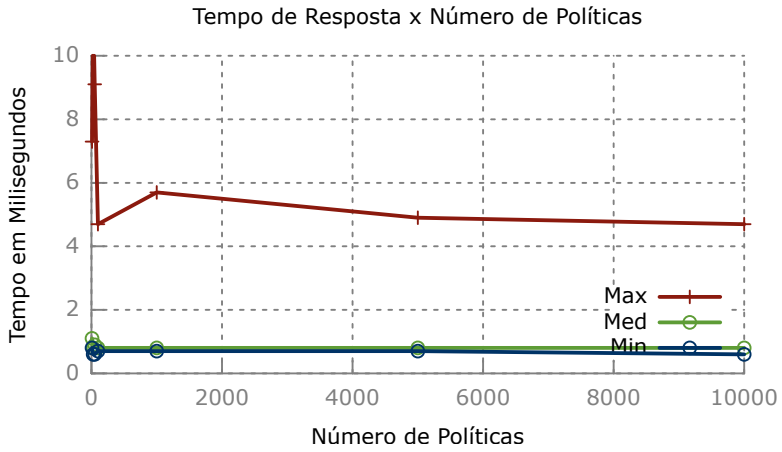

Figura 4. Avaliação: desempenho do PDP em relação ao número de políticas

Os identificadores de gabaritos de regras que deverão ser aplicados ao usuário, são transportados como se fossem um atributo da mensagem SAML. São construídos como um objeto XML que imita a estrutura hierárquica da política que deverá ser derivada no domínio local – cada fase da sessão de uso (*pre* ou *ongoing*) possui sua própria lista de identificadores, na ordem que devem ser configurados, assim como demais informações relevantes (e.g. qual o algoritmo de combinação de regras a ser usado). Essas mensagens são protegidas criptograficamente, garantindo sua autenticidade e integridade.

O domínio local, recebe as requisições contendo a credencial de autorização em um serviço web estilo *REST*, invoca um processo gerenciador de contexto feito em java, com módulos para a validação da credencial, derivação da política XACML e sua posterior avaliação. Uma *thread* é configurada com um parâmetro contido no resultado da avaliação, para periodicamente solicitar a reavaliação da política. Os dados de sessão e políticas são mantidos em um diretório local legível apenas pelo gerenciador de contexto.

B. Avaliação

O tempo de resposta do monitor de referência (PDP) em face de um número crescente de políticas armazenadas localmente foi medido (Figura 4) e considerado baixo, em média, sempre permanecendo abaixo de 2 ms. Com poucas políticas (menos que 1000), os piores tempos foram maiores que nos demais cenários, ainda assim ficaram dentro de um limite praticável de 10 milissegundos.

O segundo teste (Figura 5) demonstra a vantagem de se utilizar uma abordagem híbrida com gabaritos ao invés de configuração de políticas. Para a utilização do mesmo número de regras (controles) a abordagem de *provisioning* se mostra computacionalmente mais cara. O modelo híbrido utilizou mensagens na faixa de 4754 a 18524 bytes, enquanto o modelo *provisioning* utilizou de 11314 a 42594 bytes, num cenário envolvendo de 6 a 96 regras.

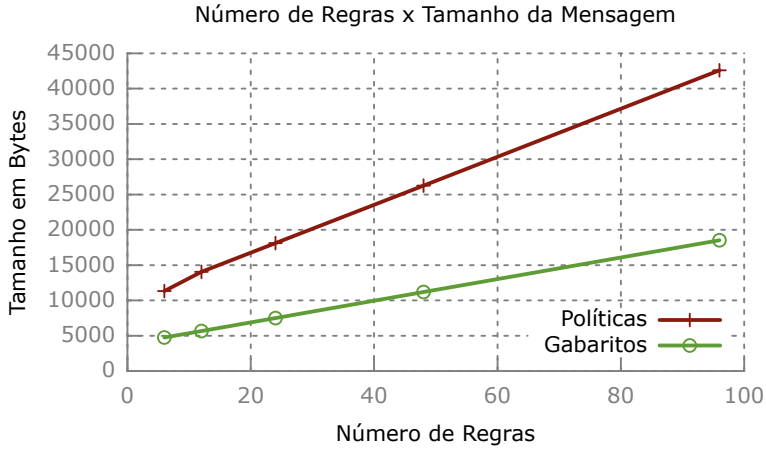

Figura 5. Avaliação: Provisioning *versus* Proposta

Os testes mostraram as vantagens da proposta em relação aos modelos tradicionais de gerência de políticas (*provisioning* e *outsourcing*), sendo que os tempos de resposta são melhores e a verbosidade das mensagens é menor. É importante ressaltar que o modelo *outsourcing* exige um protocolo do tipo requisição e resposta, que além de mais lento pode gerar uma significativa quantidade de mensagens na rede.

V. Trabalhos Relacionados

A proposta desenvolvida em [5] descreve uma arquitetura para controle de uso resiliente para computação em nuvem. O trabalho adota uma arquitetura do tipo *outsourcing* para permitir o controle de uso em um ambiente distribuído. Os autores propõem a utilização de espaços de tuplas para lidar com a alta demanda vinda dos sistemas hospedeiros, deste modo, os mecanismos de avaliação de políticas e contabilização de uso lançam novas instâncias conforme a necessidade. A resiliência da proposta consiste em permitir que discrepâncias na utilização de recursos, entre reavaliações de políticas, sejam acomodadas através de uma quota de recurso que funciona como uma margem de segurança. Entre as principais diferenças entre aquele trabalho e esta proposta estão: arquiteturas diferenciadas (híbrida de *provisioning* com credenciais de autorização e gabaritos de regras *versus outsourcing* puro e políticas estáticas); escopos diferentes (controle local *versus* controle de alto nível) e reavaliação de políticas (parametrizável *versus* fixa).

Em [6] foi descrita uma arquitetura para controle de nuvem em um ambiente IaaS. Os autores optaram por realizar extensões à linguagem XACML, com a finalidade de expressar atualização de atributos e fase de reavaliação. A arquitetura funciona em modo *outsourcing*, com um componente que monitora modificações em atributos de uso, com periodicidade pré-definida, e dispara a reavaliação de políticas de controle de uso. A possível sobrecarga dos mecanismos de avaliação de política não é abordada. Nossa proposta difere-se tanto pela abordagem arquitetural, que é híbrida, quanto pela ausência de extensões a linguagem XACML, além de oferecer reavaliação parametrizável das políticas, escopo do controle local e granularidade mais fina(aplicações *versus* máquinas virtuais).

Uma arquitetura de controle de acesso baseada no modelo UCON$_{ABC}$ é apresentada em [7]. Sua contribuição é a inclusão de um modelo de negociação, acoplado a arquitetura de autorização, flexibilizando o controle de acesso em nuvens computacionais. O sujeito tem a possibilidade de obter outra opção de acesso através de negociação, em certas situações, ao invés de ser prontamente rejeitado. O trabalho encontra-se apenas como proposta conceitual, sem demonstração de viabilidade para ambientes de nuvem ou computação distribuída.

VI. Conclusão

Este artigo apresentou e avaliou uma proposta de arquitetura para gerência distribuída de controle de uso em ambientes de PaaS, tendo como características distintivas a utilização de gabaritos de regras para derivar políticas de controle de acesso com base em credenciais de autorização. Os gabaritos são previamente armazenados nos sistemas locais, onde a avaliação da política e *enforcement* são realizados. As credenciais evitam o transporte de políticas completas através da rede, com consequente redução no tamanho das mensagens, sem contudo perder a flexibilidade na definição de políticas individuais. A sincronização de políticas é obtida através do envio de credenciais SAML com regras individuais (quando o domínio local detecta a mudança dos valores nas assertivas de autorização individuais, as políticas locais afetadas são regeradas e reavaliadas). Os testes realizados indicam que a proposta é viável, pois reduz o custo de comunicação existente em abordagens tradicionais enquanto obtém os benefícios próprios de uma abordagem de *provisioning* sem, contudo, exibir a complexidade para manter as políticas sincronizadas.

Futuramente espera-se desenvolver uma arquitetura descentralizada de compartilhamento de atributos entre domínios locais para reconfiguração dinâmica dos atributos de autorização. Isso viabilizará a cooperação do domínios locais, otimizando o uso de recursos, sem a necessidade de um sistema centralizado que exige mecanismos complexos para obter escalabilidade e robustez na presença de faltas.

Referências

[1] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, 2008.

[2] T. Mell, Peter e Grance, "The NIST Definition of Cloud Computing," *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009.

[3] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and Privacy Challenges in Cloud Computing Environments." *IEEE Security & Privacy*, vol. 8, no. 6, pp. 24–31, 2010.

[4] J. Park and R. Sandhu, "The UCONabc Usage Control Model," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 128–174, Feb. 2004. [Online]. Available: http://doi.acm.org/10.1145/984334.984339

[5] A. L. Marcon Jr., A. O. Santin, M. Stihler, and J. Bachtold, "A UCON$_{ABC}$ Resilient Authorization Evaluation for Cloud Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 457–467, Feb. 2014. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2013.113

[6] A. Lazouski, G. Mancini, F. Martinelli, and P. Mori, "Usage Control in Cloud Systems," in *IEEE International Conference for Internet Technology And Secured Transactions*, 2012, pp. 202–207.

[7] X. R. Danwei Chen, Xiuli Huang, "Access Control of Cloud Service Based on UCON," in *Cloud Computing*. Springer, 2009, pp. 559–564.

[8] S. Capizzi, "A Tuple Space Implementation for Large-scale Infrastructures," Tese de Doutorado, Università di Bologna, 2008.

[9] S. Capizzi and A. Messina, "A Tuple Space Service for Large Scale Infrastructures," in *IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2008, pp. 182–187.

[10] Red Hat, Inc., "Openshift," https://www.openshift.com/, 2015, acessado: 30-03-2015.

[11] Heroku Inc., "Heroku," https://www.heroku.com/, 2015, acessado: 30-03-2015.

[12] Pivotal Software Inc., "Cloud Foundry," http://cloudfoundry.org/index.html, 2015, acessado: 30-03-2015.

[13] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 275–287, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1272998.1273025

[14] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, "Performance Evaluation of Virtualization Technologies for Server Consolidation," *HP Labs Tec. Report*, 2007.

[15] V. Chaudhary, M. Cha, J. Walters, S. Guercio, and S. Gallo, "A comparison of virtualization technologies for HPC," in *IEEE 22nd International Conference on Advanced Information Networking and Applications, AINA.*, 2008, pp. 861–868.

[16] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith, "COPS usage for policy provisioning (COPS-PR)," Internet Requests for Comments, RFC 3084, março 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3084

[17] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, "The COPS (common open policy service) protocol," Internet Requests for Comments, RFC 2748, janeiro 2000.

[18] The FreeBSD Project, "The FreeBSD Project," https://www.freebsd.org, 2015, acessado: 30-03-2015.

[19] OASIS, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0," http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, 2014, acessado: 30-03-2015.

[20] Internet2, "Opensaml 2 for java," https://wiki.shibboleth.net/confluence/display/OpenSAML/Home, 2015, acessado: 30-03-2015.

[21] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2014, acessado: 30-03-2015.

[22] WSO2 Inc., "Balana XACML for Authorization," https://svn.wso2.org/repos/wso2/trunk/commons/balana/, 2015, acessado: 30-03-2015.