# Gerenciamento de Identidades Tolerante a Intrusões

**Luciano Barreto[1], Frank Siqueira[1], Joni da Silva Fraga[1], Eduardo Feitosa[2]**

[1]Universidade Federal de Santa Catarina
Caixa Postal 476 – 88040-900 – Florianópolis – Santa Catarina – Brasil

[2]Instituto de Computação – Universidade Federal do Amazonas

{lucianobarreto, fraga}@das.ufsc.br, frank@inf.ufsc.br,
efeitosa@icomp.ufam.edu.br

***Resumo***. *O gerenciamento de identidades é um ponto central na segurança de grandes aplicações e sistemas distribuídos. Os provedores de identidade (IdPs) são elementos que concentram informações críticas de usuários. Estas informações são armazenadas com cuidados especiais nestes provedores e invasões não necessariamente resultam em violações de segurança. Mas, intrusões com implantes de comportamentos maliciosos podem modificar a ação destes elementos de autenticação. Indivíduos não autorizados podem ser aceitos no sistema e usuários legítimos podem ter seus acessos negados. Neste artigo introduzimos uma abordagem de tolerância a intrusões que visa garantir o comportamento correto na autenticação em grandes sistemas.*

***Abstract***. *Identity management is a central point to the security of large applications and distributed systems. The identity providers are elements that concentrate critical information of users. These information are stored with special care in these providers and intrusions do not necessarily result in security violations. But intrusions may implant malicious behaviors which modify the action of these providers. In this way, unauthorized accesses may be accepted into the system and legitimate users may have their accesses denied. In this paper we introduce an approach to intrusion tolerance for ensuring the correct behavior in authentications of large systems.*

## 1. Introdução

Em sistemas distribuídos, a aplicação de políticas e a segurança de serviços do sistema dependem, principalmente, dos controles de autenticação e autorização. A abordagem convencional empregada na concretização destes controles é baseada na implementação dos mesmos sobre provedores de serviços (SPs). O usuário deve se autenticar junto a um serviço e fica sujeito à política de autorização do mesmo. A complexidade das aplicações e sistemas distribuídos atuais torna este modelo limitado. Vários modelos e infraestruturas nos últimos anos têm sido introduzidos, onde estes controles tomam forma a partir de um conceito mais amplo que é o de gerenciamento de identidades.

No caso de sistemas distribuídos de larga escala como *clouds*, grades e redes colaborativas, o gerenciamento de identidades deve se basear em uma infraestrutura que integre políticas e tecnologias, permitindo às diferentes organizações que participam destes sistemas e aplicações ultrapassar seus domínios locais (domínios administrativos ou de políticas). Diante disto, usuários e aplicações podem ter acesso, de maneira segura, a recursos remotos, desde que possuam as credenciais exigidas pelas políticas

destes recursos. A infraestrutura necessária para o gerenciamento de identidades envolve muitas vezes um sistema de autenticação (usualmente em sistemas de larga escala corresponde a uma rede de autoridades de autenticação) e um sistema de gerenciamento de atributos (serviço que fornece informações adicionais sobre usuários). Estes atributos, individual ou coletivamente, podem identificar o usuário (autenticação) e fornecer as informações necessárias (atributos do usuário) para a realização dos controles de autorização (controle baseado em atributos) permitindo assim, a execução das operações solicitadas. Em muitas propostas e infraestruturas, as autoridades de autenticação também concentram o gerenciamento de atributos e são identificados como provedores de identidades (*Identity Providers - IdP*).

Com isto, em vários modelos de gerenciamento de identidades (dentre os quais destacamos o centralizado, identidades federadas e *user-centric* [Jøsang et al. 2005]), a autenticação não é mais realizada nos SPs, mas sim em autoridades de autenticação (ou *IdPs*). A autorização continua sendo concretizada nos SPs porque estes conhecem seus recursos e políticas (de aplicação) e devem zelar pelos mesmos. Nesta situação, usuários e SPs não mais precisam manter sob sigilo listas muitas vezes enormes de credenciais (*senhas*, certificados, atributos de usuário, etc.). Estes provedores de identidades (autoridades de autenticação) passam a ser os pontos centrais na aplicação de políticas de segurança em sistemas distribuídos. Mas, também por serem serviços que ficam disponíveis via Internet, estes *IdPs* estão sujeitos a ataques que podem resultar em intrusões, o que seria catastrófico para a segurança das informações e recursos do sistema.

O trabalho que apresentamos neste texto é parte do projeto *SecFuNet*[1], que se propõe a desenvolver, dentre vários serviços, um *framework* de segurança para *clouds*, introduzindo funções de autenticação e autorização para este tipo de ambiente. Neste sentido, apresentamos neste artigo a proposta de um modelo para tornar os *IdPs* do *SecFuNet* tolerantes a intrusão. O gerenciamento de identidades do *SecFuNet* é fortemente dependente de componentes de *hardware* protegido. Os *logins* de usuários são feitos diretamente entre s*martcards* (de posse do usuário) e processadores seguros dispostos em um servidor de identidades (servidor de autenticação). No gerenciamento de identidades do *SecFuNet*, as relações de confiança entre *browsers* (usuários), *IdPs* e *SPs* fazem uso dos protocolos do *framework OpenID* [OpenID 2007].

Embora sejam empregados componentes seguros e protocolos que não deixam as informações de usuários disponíveis em um *IdP* invadido, estes cuidados não impedem que *IdPs* invadidos possam ter sido alvo de "implantações" de comportamentos maliciosos. Atuando segundo estes comportamentos, um *IdP* malicioso pode certificar indivíduos (intrusos) não autorizados por políticas, assim como também pode não autenticar usuários reconhecidos pelas mesmas políticas. As soluções para tolerância a intrusões ou a faltas bizantinas BFT ("*Byzantine Fault Tolerance"*) são normalmente baseadas em replicação Máquina de Estados (ME) [Schneider 1990]. Vários trabalhos presentes na literatura apresentam soluções práticas para tolerância a faltas maliciosas (intrusões), sendo os mais conhecidos o *PBFT* [Castro and Liskov 2002] e o *Zyzziva* [Kotla et al. 2007]. Estes trabalhos assumem que em suas replicações, enquanto os limites de faltas maliciosas (definido como *f* ) não forem atingidos, a segurança do sistema é garantida. Um dos problemas destas abordagens está no fato das mesmas usarem grande

[1] *SecFuNet Project*: "*Security for Future Networks*" Edital Brasil/Europa, MCT/CNPq número 66/2010, processo CNPq nº 590047/2011-6.

número de réplicas físicas para execução do protocolo. No *PBFT*, o número de réplicas necessárias para execução do protocolo é definido como *3f + 1*. Estes protocolos, se usados para a tolerância a intrusões de *IdPs*, devido aos altos custos em termos de mensagens, tornaria mais custoso o desempenho do procedimento de *login*.

O modelo de tolerância a intrusões que usamos neste texto é baseado na tecnologia de virtualização e não envolve replicação física o que é benéfico para o nosso sistema onde os *IdPs* fazem uso de *hardware* especializado (processadores seguros) e cuja replicação aumentaria os custos financeiros e poderiam também trazer novos problemas como o aumento da disponibilidade de informações sigilosas na rede. Nossa abordagem de replicação faz uso de memória compartilhada e é uma adaptação do modelo apresentado em [Lau et al. 2012a][Lau et al. 2012b]. As adaptações neste modelo foram produzidas para adequá-lo às características do *IdP* e aos protocolos do *framework OpenID*.

O artigo na seção 2 apresenta as características e premissas assumidas no modelo de sistema usado. Questões do esquema de tolerância a malícia são discutidas na seção 3. Na sequência, a seção 4 descreve a infraestrutura que permite a construção de provedores de identidades tolerantes a intrusões. Os aspectos de implementação de um protótipo e a apresentação de resultados de testes são focadas na seção 5. Na seção 6 são recuperados e discutidos os principais trabalhos relacionados. Por fim, na seção 7 são apresentadas as conclusões finais do artigo.

## 2. Caracterização do Ambiente de Autenticação

O processo de autenticação de usuários em sistemas distribuídos envolve, em um primeiro passo, um procedimento de *login* onde um usuário, através de trocas com um provedor de identidade (usando técnicas como *userid*/*password*, *userid*/*certificate*, *Challange*/*Response*, etc.), tem a sua identidade validada diante do sistema. Este procedimento, normalmente termina com o usuário recebendo do seu *IdP* um *token* ou asserção de autenticação que serão usados para atestar sua autenticidade diante de outras entidades do sistema em subsequentes interações.

Em sistemas de computação em nuvens, serviços de gerenciamento de identidades são usualmente implementados em *clouds* privadas ou em servidores com *hardware* especial. A não colocação de um *IdP* em *clouds* públicas se justifica pela criticidade das informações. O projeto *SecFuNet*, por enfatizar o uso de processadores seguros no processo de autenticação de usuários, implica em limitações de escala. Então o gerenciamento de identidades deve envolver vários *IdPs*, cada um com o seu próprio domínio de política. Estes provedores de identidades fornecem asserções de autenticação, seguindo suas políticas, para usuários e provedores de serviço (SPs) de seus domínios de política. Nestes domínios, portanto, o gerenciamento de identidades segue uma abordagem centralizada onde o usuário de posse de um cartão inteligente (*smartcard*), fornece informações que são verificadas em um processador seguro no *IdP* do seu domínio. O *IdP* desempenha o papel de uma terceira parte confiável nas interações entre usuário e SPs.

No entanto, para sistemas complexos como *clouds*, que normalmente se estendem além destes domínios locais, este modelo de intermediação simples é limitado. É necessário expandir esse modelo para um sistema de gerenciamento de

identidade com base em redes de confiança, envolvendo vários destes provedores locais de identidade. Ou seja, é necessário adotar abordagens de gerenciamento onde ou os *IdPs* de diversos domínio possuem relações de confiança entre si (abordagem de "identidades federadas"), ou os *SPs* possuem listas de *IdPs* em quem confiam (abordagem centrada em *SPs*). Para a aplicação destas abordagens citadas, é necessário estender a escala das relações de confiança e isto é feito com o uso de Infraestruturas de Autenticação e Autorização. Exemplos notórios destas infraestruturas são o *Shibboleth* [Lewis 2008] e o *Liberty Alliance* [Liberty 2003] para abordagens de identidades federadas e *OpenID* como abordagem centrada em SPs. No projeto *SecFuNet*, seguimos a abordagem do *OpenID*, onde cada provedor de serviços mantém uma lista de *IdPs* em que confia e que seus usuários devem ter identidades em pelo menos um destes *IdPs*.

## 3. O Modelo de Tolerância a Intrusões IT-VM

O uso da tecnologia de virtualização torna possível a implementação de novos protocolos e suportes de tolerância a intrusão. É o caso do algoritmo IT-VM (*Intrusion Tolerance by Virtual Machines*) [Lau et al. 2012a], [Lau et al. 2012b] que define réplicas de servidores implementadas em máquinas virtuais (VMs) isoladas e que se comunicam via memória compartilhada na efetivação de uma replicação Máquina de Estados (ME). As réplicas e suas VMs são mapeadas sobre somente um servidor físico neste modelo. Como o protocolo ZZ [Wood et al. 2011] o IT-VM apresenta funções para alteração dinâmica da replicação do modelo, ou seja, réplicas maliciosas quando detectadas, são removidas e substituídas de forma eficiente e sem deixar que a replicação durante seu ciclo de vida fique limitada a um número fixo de falhas no sistema.

As soluções IT-VM e ZZ operam em situações sem falhas de réplicas com replicação mínima: $f + 1$ réplicas ativas na configuração. Mas, no protocolo ZZ, as VMs de réplicas são executadas em um conjunto de máquinas físicas. Com isto, no ZZ, as VMs não interagem via memória compartilhada mesmo quando estão dispostas na mesma máquina física. Estas interações entre VMs são realizadas através de trocas de mensagens via protocolos de rede, implicando em custos adicionais no desempenho deste último protocolo. O ZZ também não trabalha com a ideia de elementos confiáveis como o modelo IT-VM e, portanto, não permite a separação de faltas *crash* e de intrusões. Como queremos desempenho e tratar somente com intrusões, o modelo IT-VM se mostram como uma base adequada para o uso nos *IdPs* do projeto *SecFuNet*.

O algoritmo IT-VM descreve diferentes componentes: i) As réplicas isoladas em diferentes VMs que são os únicos elementos visíveis via rede; ii) memória compartilhada que é usada pelos componentes em diferentes níveis para se comunicarem; iii) *Agreement Service* (*AS*), componente confiável que se mantém isolado da rede externa e das VMs e agrega funções de verificação do comportamento das réplicas assim como assinatura das respostas enviadas aos clientes. O *Agreement Service*, por executar operações vitais, é assumido como "elemento confiável" no algoritmo IT-VM. A garantia de confiabilidade do *Agreement Service* é baseada em premissas de isolamento e simplicidade deste componente, que podem facilitar a verificação e testes sobre o código do mesmo. O isolamento do *AS* e da memória compartilhada é concretizado através do *hypervisor* de virtualização.

## 4. Provedor de Identidades Tolerante a Intrusões (*IdP-IT*)

O modelo proposto para a tolerância a Intrusões do *IdP* (*IdP-IT*) no *SecFuNet* envolve a estratificação de camadas apresentada na Figura 1. O *IdP-IT* é explicitado pelo ambiente virtualizado da figura e serve de *front-end* a um servidor de autenticação (*SA*). Este servidor de autenticação é construído sobre um *grid* de processadores seguros onde cada um destes processadores mantém dados e controla as decisões de autenticação de um conjunto de usuários. O *login* é concretizado com o uso de *userid*/*certificate* onde estes certificados podem ser de uma *PKI* oficial ou formados com as chaves públicas dos usuários sendo assinadas pelo servidor de autenticação de seus domínios. As interações de *login* são realizadas com a liberação do certificado diretamente do *smartcard* do usuário. Uma vez verificada as informações de *login* de um usuário, é estabelecido um canal EAP-TLS entre o *smartcard* e o processador seguro correspondente para a troca de atributos que devem vir do cartão e ficarem disponíveis neste último durante toda a sessão de computação do usuário no sistema.

O servidor de autenticação é isolado da rede externa e dos níveis onde o nosso modelo de tolerância a intrusões atua, através de uma interface (Interface com o *SA* na Figura 1). Esta interface possui a operação *verifyCertificate*() que é usada na verificação de certificados de usuários. A operação *createSign*() é também parte desta interface e retorna uma assinatura feita com a chave privada do *SA* sobre dados passados como parâmetros. Outras operações ligadas à criação e remoção de contas de usuário, criação e remoção de certificados, introdução e remoção de atributos, etc, são visíveis apenas para administradores do servidor de autenticação.

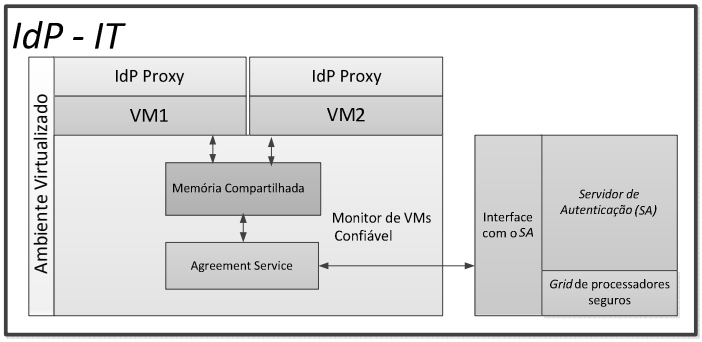


**Figura 1 – *IdP* Tolerante a Intrusões**

O trabalho descrito neste texto se fixa na parte do ambiente virtualizado da Figura 1. No nível mais alto do modelo *IdP-TI*, estão as VMs de *IdP proxies*, constituindo as réplicas do algoritmo IT-VM citado na seção anterior. As VMs devem executar os *IdPs* do *framework* definido para as relações de confiança no *SecFuNet* (no caso o *OpenID*). Estes *IdPs* na verdade não controlam as contas dos usuários, mas passam as informações obtidas dos usuários, para o servidor de autenticação que é mantido isolado do ambiente de rede (por isso identificados como proxy).

As VMs que executam os *IdPs proxies* do *OpenID* atuam como provedores diferentes com endereços próprios. Externamente são vistos como diferentes provedores de identidades *OpenID*. Suas atuações serão consideradas como corretas ou falhas diante de uma mesma requisição de *login* de um usuário. Máquinas virtuais corretas mantém a execução do protocolo segundo as especificações, enquanto máquinas virtuais faltosas podem apresentar qualquer comportamento arbitrário, tal como: adulterar mensagens; omitir seus recebimentos; omitir envios de asserções e certificar indivíduos não autori-

zados. No nível subsequente abaixo dos *IdPs proxies*, está o monitor de máquinas virtuais (VMM) que implementa a abstração de memória compartilhada, usada para a comunicação dos *IdPs proxies* e o *Agreement Service* (*AS*). O isolamento do *AS*, que é um dos requisitos para a correção do algoritmo IT-VM é garantido pela implementação do mesmo também neste nível. Ou seja, o *Agreement Service* não pode ser comprometido através de intrusões via rede. Na verdade as intrusões devem se limitar as VMs dos *IdPs proxies* que são os únicos componentes do modelo visíveis externamente.

O *AS* desempenha as seguintes funções na arquitetura da Figura 1: verificação das requisições (*OpenID)* dos clientes recebidas pelos *IdPs proxies*; comunicação via interface indicada com o servidor de autenticação, criação e assinatura da asserção *OpenID* que deve refletir o resultado do *login*. Estas asserções são mantidas no *AS* durante as seções de computação dos usuários e liberadas sob demanda a provedores de serviços segundo os protocolos do *OpenID*. Em situações normais de execução o *AS* mantém a configuração de $f$ +1 VMs de *proxies*. Na detecção de comportamento malicioso, o *AS* ativa novas $f$ VMs p*roxies* e, uma vez terminado o processamento da requisição do *login*, desativa as VMs suspeitas de comportamento malicioso, mantendo a replicação com o quórum de execução normal ($f$ +1 VMs de *proxies*).

### 4.1 Integração do *IdP* tolerante a intrusões com protocolos *OpenID*

*OpenID* é um framework para gerenciamento de identidades que segue a abordagem centralizada. Neste *framework*, o protocolo de identificação inicia com um usuário se apresentando a um provedor de serviços (chamado de *Relying Party* nas especificações *OpenID*) através de um identificador *OpenID*. Este identificador *OpenID* é usado em vários provedores de serviço (*Relying Parties*) que confiam no *IdP OpenID* gerador deste identificador. Esta confiança é necessária porque os provedores de serviços (*RPs*) devem delegar a autenticação do usuário indicado pelo identificador ao provedor de identidades (*OpenID Provider*) correspondente. Estes identificadores de usuários podem ser URLs ou ainda, documentos XRI (*Extensible Resource Identifier*) que forneçem informações adicionais como URLs de provedores de identidades onde as credenciais dos usuários possam ser aceitas.

#### 4.1.1 Descrição do protocolo de autenticação *OpenID*

O protocolo de autenticação *OpenID* é representado pela Figura 2. Este protocolo inicia com o usuário, através de seu navegador (*user-agent*), acessando o provedor de serviços (RP) (Passo1). O RP então apresenta no navegador do usuário um formulário solicitando o identificador *OpenID* do cliente (Passo2). Após a apresentação do identificador do usuário (Passo 3), é feita a normalização [Berners-Lee et al. 2005] deste identificador que tem por objetivo obter o endereço do *IdP* do usuário.

Depois de normalizado o identificador, é executado um protocolo de descoberta (Passo 4). A descoberta é um processo executado pelo RP com o objetivo de obter informações necessárias para o início das trocas de mensagem com o provedor de identidades (*IdP OpenID*). O processo de descoberta pode ser realizado através de documentos XRI ou através da URL apresentada como *ID* do usuário. O RP faz uso do protocolo *Yadis* [Miller 2006] quando a descoberta deve tratar com documentos XRI. Caso seja uma URL, então métodos HTTP são usados nesta descoberta. De posse das informações necessárias, o RP pode então estabelecer associação direta ou indireta com o *IdP* (Passo 5). A associação direta é opcional e usa troca de segredos entre as partes segundo o

protocolo *Diffie-Hellman Key Agreement*. No Passo 6, o *browser* do usuário é redirecionado para o *IdP OpenID* de modo que esse faça o seu *login*. A autenticação do usuário pode ser feita baseada no envio de sua senha (ou outro tipo de credencial) em interação com o *IdP*. Caso o usuário já tenha se identificado junto ao seu *IdP*, o RP (provedor de serviço) simplesmente solicita diretamente ao *IdP* a asserção de autenticação correspondente. O Passo 8 descreve o envio da asserção de autenticação pelo *IdP* para o RP, com informações sobre o processo de *login*. Esta asserção pode ser positiva quando o *IdP* reconhece o usuário e sua autenticação ou negativa, em caso contrário. Um dos campos importantes da asserção é a assinatura do *IdP* que garante a legitimidade do processo para o RP. No final da identificação positiva, o usuário tem o seu *browser* redirecionado para o RP, para solicitar os seus acessos seguindo as políticas de autorização do RP.

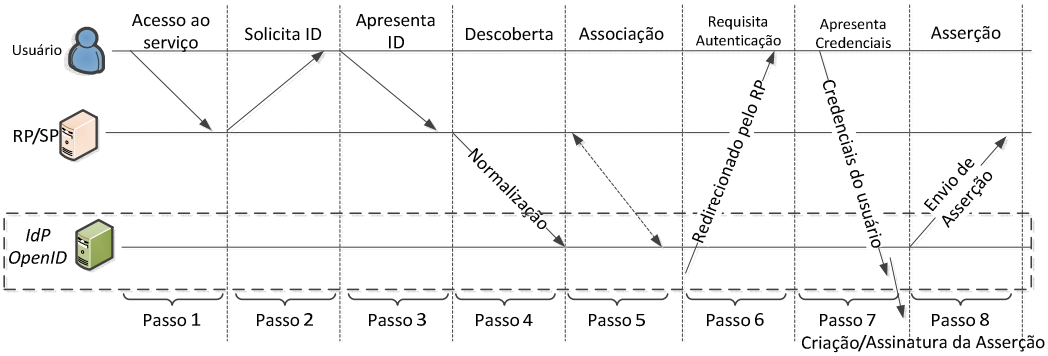


**Figura 2 – Protocolo *OpenID***

### 4.1.2 Integração de Mecanismos de TI aos Protocolos *OpenID*

Esta seção trata da integração dos protocolos e mecanismos *OpenID* e com o *IdP* tolerante a intrusões (*IdP-IT*). Esta integração é desenvolvida de modo a não resultar em modificações nos protocolos *OpenID* e de deixar transparente tanto para usuários como para provedores de serviços (RPs) a replicação usada neste *IdP*. A Figura 3 ilustra os passos do protocolo *OpenID* adaptado para a tolerância a intrusões no modelo *IdP-IT*.

O protocolo funciona de maneira similar ao protocolo *OpenID* apresentado na seção 4.1.1 até o Passo 4. Como a abordagem de replicação conta sempre com mais de um servidor (pelo menos *f+1* servidores), o passo de descoberta deve ser feito obrigatoriamente utilizando o protocolo de descoberta *Yadis*, que retorna para RPs descritores indicando provedores de identidade. Quando um cliente apresenta um identificador *Yadis* a um provedor de serviços, este identificador indica para o RP uma URL onde pode ser liberado um *Yadis Document* com descritores para *IdPs*. Geralmente são encontrados neste documento todos os provedores de identidade que podem ser utilizados para autenticar um usuário. Esta lista contendo os descritores pode definir prioridade de uso dos recursos (no nosso caso *IdPs*) indicados por estes descritores.

No protocolo proposto nesta seção, é importante sempre que sejam retornados todos os descritores dos *IdP proxies* (os *IdPs OpenID* replicados) presentes no modelo *IdP-IT*. Para não modificar o protocolo *OpenID* nas interações *browser* / *IdPs*, foi necessário adaptar o modelo IT-VM que inicialmente previa replicação ativa, para replicação passiva (sem estado). Nesta nova situação, o *IdP proxy* no papel de primário deve interagir com o *browser* e o SP. E caso este *proxy* apresente comportamento malicioso, outra réplica ativa (um *IdP proxy* no papel de *backup*) passa a assumir a sua

função de primário. Na lista de descritores retornada ao RP na descoberta com o *Yadis*, os diferentes *proxies* da replicação de *IdPs* deverão ser sinalizados com diferentes prioridades indicando os seus papéis (primário ou *backups*). Assumindo então diferentes papéis, o SP deve estabelecer a associação do Passo 5 com o *proxy* primário.

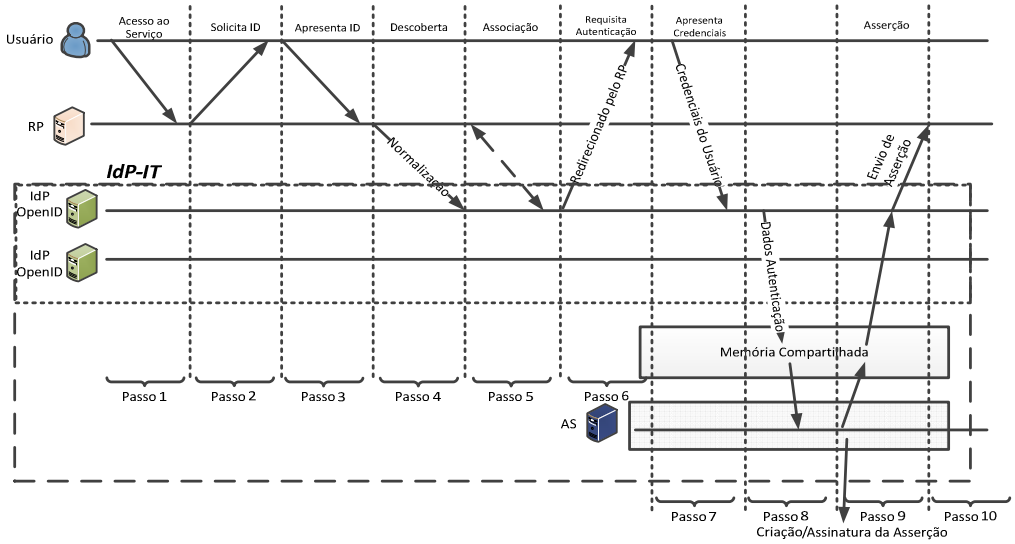


**Figura 3– Protocolo *OpenID* estendido para tolerância a intrusões**

O *proxy* primário recebe então a credencial do usuário (certificado) em mensagem *OpenID* (Passo 7), extrai então esta credencial e escreve a mesma na memória compartilhada para que seja lida pelo *Agreement Service* (passo 8). O *Agreement Service* deve então submeter o certificado do usuário às verificações correspondentes, usando a operação *verifyCertificate*() da interface do servidor de autenticação subjacente (seção 4 e Figura 1). Se o retorno desta operação indicar *true*, a credencial está correta e o *Agreement Service* pode criar a asserção de autenticação correspondente ao *login* do usuário (Passo 9, Figura 3).

A asserção tem informações como a identidade do usuário, a URL para a qual o *IdP proxy* no papel de primário deverá redirecionar o *browser* do usuário, um *handle* da associação *RP-IdP* (*proxy* primário), o *nonce* e a assinatura do servidor de autenticação sobre todos estes campos. A assinatura destas informações é conseguida pelo *Agreement Service* usando a operação *createSign*() da interface do servidor de autenticação. Para completar o Passo 9, o *Agreement Service* deve passar esta asserção para o *proxy* primário para que este, por sua vez, envie através do protocolo *OpenID* esta asserção ao provedor de serviço (RP). A chave pública do servidor de autenticação, na forma de um certificado auto assinado (ou assinado por uma chave disponível hierarquicamente superior dentro de uma *PKI*) tem que estar disponível para os RPs participantes das interações. Feitas as verificações (assinaturas e demais informações) sobre a asserção pelo RP, este pode conceder o acesso requisitado ao usuário, que ficará sujeito também a verificações de políticas de autorização locais. Vale ressaltar que no protocolo *IdP-IT*, mesmo envolvendo a inclusão de replicação (*IdP proxies*), nenhuma modificação nas mensagens *OpenID* foi necessária.

### 4.1.3 Execuções anormais do protocolo *IdP-IT*

Porém, apesar de todo o isolamento das entidades críticas, ataques aos *IdPs proxies* não podem ser evitados. Na abordagem proposta, quando as réplicas são compro-

metidas por intrusões, não há uma grande preocupação quanto a possíveis modificações de asserções ou mesmo o envio de credenciais falsas, uma vez que estas informações estão sujeitas a assinaturas ou a verificações envolvendo a parte confiável do *IdP-IT* que é o servidor de autenticação (isolado e livre de intrusões). Porém, os atacantes podem corromper réplicas de *IdP proxies* de modo que estas não executem corretamente o protocolo *OpenID*. Os mecanismos usados em conjunto com replicação de *proxies* devem detectar possíveis ações maliciosas e remover o *proxy* comprometido do *IdP-IT*.

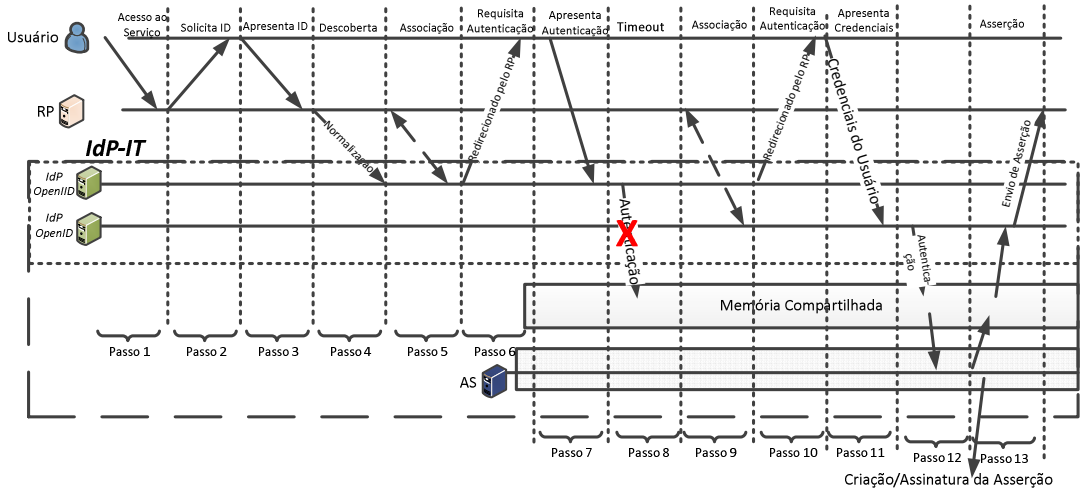


**Figura 4 – Protocolo *OpenID* replicado na presença de intrusões**

A Figura 4 ilustra o comportamento do protocolo *IdP-IT* estendido em presença de malícia. Os passos 1 a 7 ocorrem da mesma forma como apresentado na seção 4.1.2. Lembrando que no passo 4, onde é executado o protocolo de descoberta *Yadis*, é retornada para o RP a lista de *proxies* do *IdP-IT* que este pode fazer uso. Porém, na instância de protocolo apresentada na Figura 4, é mostrado o comportamento errôneo do *proxy* primário da replicação que se recusa a inserir a credencial (certificado) enviada pelo usuário na memória compartilhada, impedindo que o protocolo possa prosseguir na sua execução normal.

A detecção desta anomalia é conseguida com *timeout* no RP que solicitou a autenticação do usuário. Dessa forma, o RP ativa seu temporizador aguardando uma asserção de autenticação como resposta quando iniciou qualquer processo de troca de mensagem com o *IdP proxy* primário. Se estas não chegam dentro de prazo determinado, o RP utiliza-se da lista de descritores retornados pelo *Yadis*, para tentar conexão com outro *proxy* do *IdP-IT*. Na Figura 4, com o temporizador do RP excedido, este redireciona o pedido de autenticação do usuário para o *proxy backup* (Passo 9) no sentido que apresente sua credencial (passo 10). Este processo pode se repetir *f* vezes até que se atinja uma réplica correta (com *f* substituições de *proxies* no papel de primário) e então o usuário é autenticado. Todas as situações de comportamentos maliciosos (recusa de recebimento e envio de dados e associações, modificações em asserções, etc) são detectadas a partir do RP (provedor de serviços) com temporizações e verificações locais.

Além das considerações citadas, a configuração *f + 1* réplicas *proxies* é sempre restaurada, de modo que o *IdP-IT* mantém sempre no seu ciclo de vida o mesmo número de réplicas. As réplicas maliciosas são sempre removidas e substituídas por novas réplicas *proxies*. Estas substituições são muito simples porque os *proxies* não mantêm informações de estado. As informações de estado (informações de conexões, asserções e

demandas de autenticação) estão contidas no *Agreement Service* que é mantido protegido, isolado da rede externa. Mas, as reconfigurações geram algumas implicações: o documento retornado pelo protocolo *Yadis* deve ser modificado para a próxima demanda de autenticação de um *RP*. Ou seja, os novos *proxies* deverão possuir descritores e também prioridades indicando seus papéis nesta nova configuração no documento *Yadis*. O *Agreement Service* é o agente destas reconfigurações tratando de ativações e remoções de *proxies*. É também este serviço que prepara o documento *Yadis* com as alterações de configuração no *IdP-IT*.

## 5. Implementação de Protótipo e Análise de Resultados

De forma geral, a estrutura utilizada em nosso protótipo é representada pela Figura 5. Para a replicação dos *IdP proxies* foi utilizado o software de virtualização XEN, para a execução das máquinas virtuais. Além disso, o uso do *hypervisor* XEN oferece a facilidade de implementação da memória compartilhada para a comunicação entre as VMs *proxies* e o *Agreement Service*. A memória compartilhada e o *Agreement Service* foram implementados no nível do *hypervisor*, o que permitiu o isolamento destas abstrações, de forma que ambas possam ser consideradas componentes confiáveis, que é premissa básica da proposta. Nas VMs de *proxies* foram utilizadas diferentes versões do sistema operacional Linux, com o objetivo de manter certa independência de vulnerabilidades já que estas máquinas são os únicos componentes do *IdP-IT* visíveis externamente (via rede).

Como provedor de identidades (*OpenID*) foi utilizada a implementação em código livre *JOID*, que foi alterada para que o protocolo fosse executado diante das características de replicação e uso de componentes confiáveis, como descritas nas seções anteriores. Vale salientar que foram removidos da implementação *JOID* os módulos de gerenciamento de conta de usuários. O provedor de identidades *OpenID* implementado é executado sobre o servidor de aplicação Apache *Tomcat*. O *Agreement Service* e outras entidades introduzidas pelo modelo *IdP-IT* foram desenvolvidos com a linguagem de programação Java.

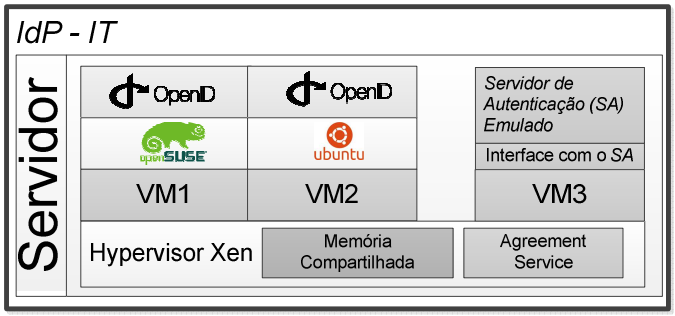


**Figura 5 – Estrutura do sistema**

O provedor de serviços (*Relying Party*) desenvolvido não precisou de extensões especiais em relação ao protocolo *OpenID*, mas este tem que saber lidar com a indisponibilidade momentânea de um IdP *proxy*. Diante disto, este RP deve contatar outro *proxy* (como alternativa do *IdP-IT*). Fizemos uso da tecnologia AJAX para o desenvolvimento do provedor de serviços. O uso do AJAX nos permitiu que houvesse um maior controle sobre a aplicação e suas trocas de mensagens com o *IdP-IT*. Ainda que ocorram trocas de provedor de identidades (na verdade trocas de *IdPs proxies*), toda a interação é feita pela aplicação executada no navegador do cliente. O usuário não precisa começar

todas as trocas de novo a partir do RP. Na substituição de *IdP* primário, a página de *login* do *OpenID* é reativada em seu *browser*.

O servidor de autenticação, mantido completamente isolado da rede, é quem realmente gerencia as contas de usuários e está sendo desenvolvido no *SecFuNet* usando um *grid* de microcontroladores seguros[2], cada um desses com *software* embutido. A estrutura geral do servidor de autenticação é concretizada sob um sistema operacional dedicado. Como o foco deste texto é o ambiente virtualizado que deve tolerar intrusões e interagir com o servidor de autenticação, emulamos o comportamento deste servidor a partir de uma máquina virtual isolada. O protótipo final do projeto *SecFuNet* prevê o *IdP-IT* e todos os seus componentes (incluindo também o servidor de autenticação), compartilhando um mesmo servidor físico em seu domínio de políticas.

### 5.1 Experimentação do Protótipo

Com o objetivo de testar nossas proposições o protótipo foi executado em um servidor com processador Intel Core i7 3.4 *Ghz*, com 8 GB de memória conectado com seus clientes através de uma rede local Ethernet funcionando a 100 *Mbps*. Em nossos testes usamos o envio automático de dados de identificação a partir de uma máquina cliente. Definimos o limite de réplicas maliciosas como sendo o valor de $f = 1$. Desta forma, como a arquitetura dispõe de $f + 1$ réplicas *proxies* executando o *IdP OpenID*, o total de réplicas usado no protocolo foi 2. O valor escolhido para $f$ não tem muito significado em relação à robustez do modelo porque, imediatamente a detecção, esta réplica corrompida é removida e ativada outra em seu lugar. Os reflexos no desempenho não são muito sentidos porque a iniciação de novos *proxies* não envolve transferências de estado.

Realizamos testes envolvendo três rodadas de 1000 operações de autenticação. Estes testes foram divididos em duas classes: i) execução de autenticações sem falhas maliciosas ii) execução de autenticações com falhas maliciosas nos *proxies*. Na primeira classe de testes (sem falhas) as simulações não levaram em conta a presença de intrusões no sistema, de forma que o serviço funcionasse sempre sem a necessidade da troca do *proxy* primário. Na segunda classe, foram incluídas modificações no sistema de forma que intrusões fossem simuladas. Nessas simulações a classe de faltas (maliciosas) foi no domínio de valores. Ou seja, os proxies maliciosos alteravam asserções de forma que quando checadas pelo RP fossem verificadas como inválidas.

Como resultado dos testes, foi calculada a média dos tempos de resposta nas identificações. Visto que em nossa abordagem a replicação é renovada, montamos casos de testes com diferentes distribuições de réplicas maliciosas. As distribuições são apresentadas na Tabela 1. A coluna $\mathbf{f}_{\%}$ apresenta, dentro do conjunto de 1000 requisições, qual a porcentagem de falhas maliciosas que ocorrem no conjunto. A coluna $\mathbf{N}_{f}$ apresenta o número de vezes que as falhas ocorrerão. Na Figura 6 é apresentado o tempo médio de autenticação nas duas classes de testes citadas acima. No caso de teste onde não existem *proxies* maliciosos, o tempo médio foi de 7 *ms*. Nos demais testes, onde as intrusões foram simuladas, este tempo foi crescente em relação ao aumento do número de intrusões. Este crescimento está relacionado ao tempo de renovação da replicação no

[2] As placas com o *grid* de microcontroladores seguros formam um produto de propriedade intelectual das empresas Implementa, EtherTrust e Telecom ParisTech.

algoritmo. No nosso esquema de testes temos 2 réplicas participando da configuração de *IdP proxies* (*f = 1*). Também para evitar os custos de criação de novas réplicas (em torno 1 segundo), mantemos 8 réplica (VMs com os seus proxies) em espera fora da configuração. A remoção de uma faltosa envolve a ativação destas réplicas em espera. Vale a pena salientar que o teste com 100% de falhas reflete que, a cada requisição de autenticação ocorra uma intrusão.

**Tabela 1 – Distribuição das faltas**

| $f_{\%}$ | $N_f$ |
|---|---|
| **0,1%** | 1 |
| **1,00%** | 10 |
| **2,50%** | 25 |
| **5,00%** | 50 |
| **10,00%** | 100 |
| **12,50%** | 125 |
| **16,60%** | 166 |
| **25,00%** | 250 |
| **50,00%** | 500 |
| **100,00%** | 1000 |

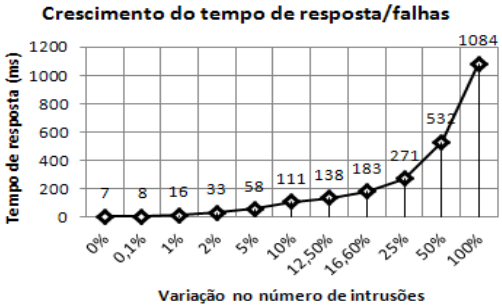


**Figura 6 – Tempo de resposta das autenticações**

Apesar de não haverem comparações com outras abordagens de forma a atestar a eficiência do algoritmo proposto, os testes serviram para demonstrar que a abordagem é factível e como esta se comporta em um ambiente hostil onde intrusões são frequentes.

## 6. Trabalhos relacionados

Até o presente momento não foram encontradas na literatura abordagens que tratassem de tolerância a intrusões em provedores de identidades. Porém, o uso de provedores de identidade vem sendo crescente em aplicações e sistemas distribuídos de larga escala como computação em nuvem, organizações virtuais, computação em grade, etc. Alguns trabalhos citados nesta seção são confrontados ou mesmo usados como mecanismo de apoio no sentido de evidenciar resultados de nossa proposta.

Em [Coppola et al. 2012] é descrito um infraestrutura para federação de provedores de computação em nuvem chamada *Contrail*. Esta infraestrutura é responsável pelo gerenciamento dos recursos oferecidos por nuvens federadas. Dessa forma, quando o cliente deseja implantar sua aplicação em nuvem, o *framework* citado seleciona a nuvem que melhor possa atender aos requisitos de *SLA* (*Service Level Agreement*) negociados com o cliente. Do ponto de vista do gerenciamento de identidades, ainda que as aplicações de um cliente estejam sendo executadas em várias nuvens distintas, este não deve necessariamente se apresentar com uma identidade diferente em cada um dos provedores das mesmas. Desta forma, na abordagem *Contrail* é definido um nível de gerenciamento de identidades que estabelece relações de confiança entre os provedores de identidade usados nas diferentes nuvens. Estas relações de confiança definem então uma federação de nuvens. Para tanto, foi utilizado nesta infraestrutura o protocolo *OAuth2*.0 [Microsoft 2012] para a geração de *tokens* de autenticação e de autorização e documentos SAML (*Security Assertion Markup Language*) para troca de certificados entre os *IdPs* de diferentes nuvens. Os *IdPs* desta proposta são implementados nas próprias nuvens, o que certamente pode ser um obstáculo significativo para se garantir a segurança dos mesmos. Dentre os trabalhos futuros propostos pelos autores está a integração do protocolo *Shibboleth* para identificação dos usuários.

Outro trabalho que propõe o uso de federação a partir de *IdPs* de diferentes nuvens é apresentado em [Tusa et al. 2012]. Neste trabalho são discutidos principalmente detalhes da implementação de autenticação única (*Single Sign On*) através da integração do protocolo *Shibboleth*, SASL (*Simple Authentication and Security Layer*) e SAML (*Security Assertion Markup Language*). Na abordagem apresentada, quando um provedor de computação em nuvem (chamado de Origem) deseja utilizar recursos de outra nuvem (chamado de Destino) este inicia o processo com trocas de mensagens através do protocolo XMPP. Esta proposta também implementa os provedores de identidades nas correspondentes nuvens e, portanto, sem os cuidados de segurança necessários.

Em [Leandro et al. 2012] é proposta uma arquitetura de federação de provedores de identidades para o uso em computação em nuvem, utilizando o protocolo *Shibboleth*. Neste trabalho é apresentada ainda uma implementação da arquitetura proposta. Nessa abordagem, os clientes possuem seus provedores de identidades fora da nuvem. Cada provedor de identidades define um domínio de políticas com os seus clientes e provedores de serviços (provedores de computação em nuvem). A associação de confiança que garante a transposição de uma asserção SAML de autenticação de um domínio para o outro é estabelecida pelo *Shibboleth*.

Os trabalhos citados acima tratam o gerenciamento de identidade de maneira similar, com o uso do conceito de federação. Somente a última experiência citada é que constrói os provedores de identidades fora das nuvens. A nossa experiência descrita neste texto, não usa o conceito de identidades federadas. O nosso modelo é o centralizado em provedores de serviço (que na verdade é a abordagem da ferramenta *OpenID*), ou seja, cada serviço (RP) mantém uma lista de provedores que este confia e localiza em qual destes o usuário está registrado usando o identificador *OpenID* do usuário. Uma das vantagens do *OpenID* é que este não requer nenuma um *pilha* de protocolos especiais do lado do cliente. O que não é o caso do *Shibboleth* que requer um *stack* de protocolos no lado do cliente, limitando com isto o seu uso em determinados dispositivos móveis. Em relação ao modelo *IdP*-TI acreditamos ser uma experiência única neste nível. Possíveis intrusões que resultem em comprometimento de *IdPs* (*proxies* no nosso modelo), não resultam na quebra de confidencialidade de dados de usuários e muito menos na interrupção do serviço de autenticação de usuários.

## 7. Conclusão

Este artigo descreveu a nossa experiência no desenvolvimento de provedor de identidades *OpenID* tolerante a intrusões. A arquitetura que implementamos faz uso da tecnologia de virtualização e não envolve replicação física do *hardware* especializado (processadores seguros) que é usado nos *IdP*. As informações de usuários são mantidas em compartimentos separados e qualquer intrusão no modelo se traduz em ações inócuas. A separação dos compartimentos é feita através do isolamento das máquinas virtuais e componentes confiáveis, que se comunicam exclusivamente através de uma memória compartilhada.

O uso da virtualização permite a recuperação da replicação definida já a partir da detecção do comportamento malicioso. Um protótipo foi implementado do modelo desenvolvido e os testes realizados com os mesmos nos dão a certeza da viabilidade das nossas propostas. Este trabalho nos parece singular porque não encontramos experiências correlatas na literatura. As propostas e desenvolvimentos apresentados neste texto fazem parte do Projeto *SecFuNet*.

## Agradecimentos

Este trabalho foi financiado pelo CNPq através dos processos 590047/2011-6 (Projeto SecFuNet) e também pelo processo 307588/2010-6. Agradecemos ainda a CAPES pelo apoio financeiro com bolsa de doutorado.

## Referências

Berners-Lee, T., Fielding, R. and Masinter, L. (2005). [RFC 3986] Uniform Resource Identifier (URI).

Castro, M. and Liskov, B. (2002). Practical Byzantine Fault Tolerance and Proactive Recovery. ACM Transactions on Computer Systems, v. 20, n. 4, p. 398–461.

Coppola, M., Dazzi, P., Lazouski, A., et al. (2012). The Contrail approach to cloud federations. The Internation Symposium on Grids and Clouds (ISGC) 2012.

Jøsang, A., Fabre, J., Hay, B. and Pope, S. (2005). Trust requirements in identity management. Proc. Australasian workshop on Grid computing and e-research, p. 99–108.

Kotla, R., Alvisi, L., Dahlin, M., Clement, A. and Wong, E. (2007). Zyzzyva: speculative byzantine Fault Tolerance. ACM SIGOPS Operating Systems Review, v. 41, n. 6, p. 45–58.

Lau, J., Barreto, L. and Fraga, J. (2012a). An Infrastructure based on Virtualization for Intrusion Tolerant Services. 19th IEEE International Conference on Web Services, p. 170 - 177.

Lau, J., Barreto, L. and Fraga, J. (2012b). Infraestrutura Baseada em Virtualização para Serviços Tolerantes a Intrusões. XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, p. 522 - 535.

Leandro, M. A. P., Nascimento, T. J., Santos, D. R. Dos, Westphall, C. M. and Westphall, C. B. (2012). Multi-Tenancy Authorization System with Federated Identity for Cloud-Based Environments Using Shibboleth. ICN 2012, The Eleventh International Conference on Networks, n. c, p. 88–93.

Lewis, J. A. (2008). Authentication 2.0 - new opportunities for online identification.

Liberty (2003). Introduction to the Liberty Alliance Identity Architecture.

Microsoft (2012). [RFC 6749] The OAuth 2.0 Authorization Framework.

Miller, J. (2006). Yadis Specification 1.0.

OpenID (2007). OpenID authentication 2.0.

Schneider, F. B. (1990). Implementing Fault-Tolerant Approach: A Tutorial Services Using the State Machine. Computing, v. 22, n. 4.

Tusa, F., Celesti, A., Villari, M. and Puliafito, A. (2012). Federation Between CLEVER Clouds Through SASL/Shibboleth Authentication. The Fourth International Conference on Evolving Internet, n. 4, p. 12–17.

Wood, T., Singh, R., Venkataramani, A., Shenoy, P. and Cecchet, E. (2011). ZZ and the Art of Practical BFT Execution. Proceedings of the sixth conference on Computer Systems EuroSys ’11, p. 123–138.