Um Modelo Funcional para Serviços de Identificação e Autenticação Tolerantes a Faltas e Intrusões

Diego Kreutz[1], **Eduardo Feitosa**[2], **Oleksandr Malichevskyy**[1]
Kaio R. S. Barbosa[2], **Hugo Cunha**[2]

[1]LaSIGE/FCUL, Universidade de Lisboa, Portugal
{kreutz,olexmal}@lasige.di.fc.ul.pt

[2]Instituto de Computação, UFAM, Manaus, Brasil
{efeitosa,kaiorafael,hugo.cunha}@icomp.ufam.edu.br

Abstract. *The correct and continuous operation of identity providers and access control services is critical in new generations of networks and online systems, such as virtualized networks and on-demand services of cloud computing. In this perspective, this paper proposes and demonstrates a functional model for architectures of identification and authentication services that are fault- and intrusion-tolerant. The feasibility and applicability of the model are evaluated through prototypes implemented for OpenID and RADIUS services. The results and analysis indicate that a functional model for prototyping and deploying more resilient and reliable identification and authentication services is feasible.*

Resumo. *A operação correta e contínua de provedores de identidade e serviços de controle de acesso são pontos críticos em novas gerações de redes e sistemas online, como redes virtualizadas e serviços sob demanda da computação em nuvem. Nesta perspectiva, este artigo tem como objetivo propor e demonstrar um modelo funcional para arquiteturas de serviços de identificação e autenticação tolerantes a faltas e intrusões. A viabilidade e aplicabilidade do modelo são avaliadas através de protótipos concretizados para os serviços OpenID e RADIUS. Os resultados indicam que um modelo funcional para o desenvolvimento de serviços de identificação e autenticação mais resilientes e confiáveis é viável.*

1. Introdução

A proliferação de provedores de infraestrutura de autenticação e autorização (AAI) vem aumentando e grande parte desse movimento se deve a necessidade de permitir que usuários acessem diferentes serviços (e.g. Facebook, Google, Twitter, Amazon) sem precisar efetuar várias autenticações ou usar credenciais específicas (uma por serviço).

Tipicamente, provedores de AAI empregam provedores de identidade e protocolos de autenticação e autorização para identificar o usuário e fornecer as informações necessárias para autorizar o acesso ao recurso ou sistema. Embora sejam serviços comumentemente essenciais para suportar controle de acesso em infraestruturas de redes, *clouds* e *grids*, a resiliência e a confiabilidade desses provedores ainda representam questões em aberto. Isto é algo preocupante com o crescente aumento dos incidentes causados por ataques digitais [Verizon RISK Team 2013]. Adicionalmente, ameaças avançadas persistentes [Tankard 2011] estão ganhando força e tornando-se um dos centros de atenção das equipes de segurança, instituições e governos.

Em termos práticos, a maioria dos serviços autenticação, autorização e *accounting* (AAA), baseados em RADIUS [Rigney et al. 2000], e provedores de identidade, basea-

dos em OpenID [Recordon and Reed 2006], não consideram certas propriedades ou recursos de segurança e dependabilidade, como pode ser observado em [FreeRADIUS 2012, RADIUS Partnerships 2008, Juniper Networks 2010, OpenID 2010, Clamshell 2013]. Em alguns casos (implementações), apenas conexões SSL e esquemas simples de replicação (*primary-master*) para tolerar falhas por paragem são utilizados. Consequentemente, ainda há espaço para investigação e desenvolvimento de soluções mais resilientes e confiáveis, capazes de lidar com os novos desafios pela frente, como ataques mais frequentes e avançados.

Neste contexto, este trabalho propõe o primeiro modelo funcional para arquiteturas de serviços de identificação e autenticação tolerantes a faltas e intrusões. A aplicabilidade do modelo é demonstrada através de dois casos de uso, como prova de conceito. O artigo discute os componentes, modelos de sistema e blocos de construção fundamentais no desenvolvimento de serviços tolerantes a faltas e intrusões. Complementarmente, são analisados aspectos sobre as vantagens e desvantagens de diferentes abordagens de implementação e implantação, como a utilização de uma *versus* múltiplas máquinas físicas.

O artigo está estruturado da seguinte forma. Na seção 2 é apresentada a motivação do trabalho. O modelo funcional é discriminado e discutido na seção 3. Finalmente, a seção 4 discute os resultados e a seção 5 apresenta as considerações finais.

2. Motivação

Serviços resilientes e confiáveis estão se tornando partes essenciais de qualquer sistema de computação ou infraestrutura de rede que necessite de características como elasticidade, disponibilidade, escalabilidade e confiabilidade. Sob esta perspectiva, os provedores de identificação e autenticação são exemplos da falta de soluções com propriedades de resiliência e confiabilidade mais fortes.

Soluções de identificação e autenticação são comumente baseadas em protocolos como RADIUS [Rigney et al. 2000] e o OpenID [Recordon and Reed 2006]. O primeiro fornece autenticação, autorização e contabilidade para infraestrutura de redes, como redes institucionais, provedores de comunicação de dados DSL/ADSL e redes GSM/3G/4G. O segundo é utilizado para autenticar usuários de serviços online em provedores de identidade, permitindo uma única identificação para o acesso a diferentes serviços, em outras palavras, tornando mais praticável sistemas *Single Sign-On*.

Contudo, ambos os protocolos não foram projetados com características robustas de segurança (e.g. confidencialidade) e dependabilidade, sendo alvos frequentes de ataques e tentativas de furto de dados (e.g. credenciais de usuários). O RADIUS apresenta um conjunto de vulnerabilidades decorrentes da própria especificação ou má implementação [Feng 2009]. Em relação a falhas no protocolo, o RADIUS não valida a integridade de alguns pacotes (`Access-Request`), usa o algoritmo `MD5hashing` para criptografar atributos e não fornece prevenção contra ataques de reflexão (*replay*). Já na implementação, o RADIUS é suscetível a ataques de dicionário, *man-in-the-middle*, *spoofing*, entre outros. Por sua vez, o OpenID também possui uma série de problemas. Pesquisas recentes mostram que ele é vulnerável a ataques de *cross-site request forgery* [Sun et al. 2012], além de *phishing*, *man-in-the-middle*, *replay*, DoS e manipulação de parâmetros do protocolo [van Delft and Oostdijk 2010, Uruena et al. 2012].

Finalmente, existe o fato do número de ameaças e ataques estar em crescimento

contínuo e o surgimento de novas ameaças avançadas persistentes [Tankard 2011] traz preocupações adicionais. Hoje, a maioria dos serviços de rede não está preparada para assegurar uma correta operação em circustâncias adversas, como aquelas representadas por falhas de infraestrutura local, ataques e ameaças avançadas.

Embora existam possíveis soluções para melhorar a segurança desses serviços, como o uso de técnicas avançadas de replicação, virtualização e componentes seguros (e.g. *smart cards* e TPMs), essas abordagens são complexas ou utilizadas de forma isolada. Um exemplo é a utilização de TPMs para criar variantes confiáveis de sistemas de gerenciamento de identidades [Leicher et al. 2010]. Embora a solução permita criar serviços OpenID confiáveis, ela não suporta propridades como alta disponibilidade e tolerância a faltas ou intrusões no serviço como um todo, focando apenas na confiança da emissão dos tickets. Outro examplo é uma infraestrutura de coordenação cooperativa de serviços Web [Alchieri et al. 2008], que propõe mecanismos de segurança que permitem uma coordenação confiável mesmo na presença de componentes maliciosos. A solução proposta atua como infrastrutura integradora de serviços Web, contando com um conjunto de gateways de serviços e um espaço de tuplas resiliente. Embora a solução ofereça tolerância a faltas e intrusões, não representa um modelo geral para serviços de rede e nem oferece recursos como componentes seguros para garantir a confidencialidade de dados sensíveis. Isso evidencia a falta de modelos e arquiteturas de sistemas que sirvam de suporte ao desenvolvimento de serviços de identificação e autenticação seguros e resilientes.

A principal motivação para do trabalho é a falta de modelos funcionais, técnicas e metodologias para o suporte ao desenvolvimento de serviços tolerantes a faltas e intrusões. Estes serviços, ao invés de resolver todos os problemas existentes, partem do pressuposto que falhas, ataques e intrusões irão eventualmente ocorrer. Sendo assim, utilizam mecanismos capazes de manter a correta operação do serviço mesmo em casos de intrusões. O modelo proposto é analisado e discutido a partir de dois protótipos.

3. Modelo Funcional

As seções seguintes apresentam uma visão geral do modelo funcional, o modelo estendido (com mecanismos de resiliência), os principais blocos de construção para concretização do modelo, propriedades e cenários de uso.

3.1. Visão Geral

A Figura 1 ilustra os componentes básicos do modelo funcional. Os quatro principais elementos são: (a) cliente; (b) serviço alvo; (c) *gateway*; e (d) Serviço Tolerante a Faltas e Intrusões (STFI). Adicionalmente, o modelo permite o uso de componentes seguros, em qualquer um dos elementos, para fornecer propriedades/garantias adicionais de temporização e/ou segurança a partes críticas da solução.

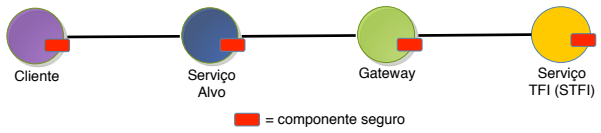

Figura 1. Modelo funcional

Um cliente pode ser um usuário tentando acessar a rede ou um dispositivo de rede que precisa consultar um controlador para tomar alguma decisão sobre um novo evento

registrado. Já um serviço alvo é um elemento genérico. Em uma infrastrutura típica de rede ele pode representar componentes como roteadores sem fio ou comutadores de acesso Ethernet. No caso de uso do OpenID, o serviço alvo pode ser praticamente qualquer coisa, indo desde aplicações de gestão da rede até sub-sistemas de controle de acesso em sites de comércio eletrônico.

Na sequência, o *gateway* é um elemento de propósito mais específico. Sua funcionalidade primária é prover conexão entre o serviço alvo e o serviço tolerante a faltas e intrusões. Consequentemente, este elemento precisa compreender diferentes protocolos, atuando de forma similar a um *gateway* de rede entre STFIs, serviços alvo e clientes. Uma segunda atribuição do *gateway* é mascarar os protocolos de replicação e mecanismos utilizados para implementar STFIs.

Já o STFI representa o serviço crítico da infraestrutura, com requisitos de segurança e dependabilidade mais elevados. Um dos pressupostos é que estes serviços devem tolerar diferentes tipos de faltas, como as Bizantinas (faltas arbitrárias, como as causadas por comportamentos imprevistos ou ataques). Adicionalmente, o funcionamento correto e seguro do serviço deve ser assegurado mesmo no caso de intrusões. Serviços AAA, provedores OpenID, sistemas de monitoramento e controladores de rede são exemplos de sistemas críticos em infraestruturas de rede. Qualquer falha em um desses serviços pode ter um grande impacto na operação da rede e/ou dos outros serviços, com potencial impacto negativo a usuários e negócios.

Por fim, o componente seguro pode ser utilizado em qualquer um dos elementos do modelo para garantir propriedades ou funcionalidades adicionais, como confidencialidade, verificações de integridade e garantias temporais para partes específicas dos outros elementos. Por exemplo, as chaves dos usuários podem ser armazenadas em *smart cards*. Similarmente, chaves de autoridades certificadoras (CAs) locais e de servidores podem ser também armazenadas dentro de componentes seguros. Adicionalmente, todas as operações criptográficas críticas, que utilizam o material sensível armazenado de forma segura, devem ser executadas dentro do componente seguro, sem comprometer a confidencialidade dos dados mesmo na presença de um invasor.

3.2. Agregando Tolerância a Faltas e Intrusões

Como pode ser observado na Figura 2, a arquitetura geral do modelo funcional assume a existência de diferentes limiares e/ou técnicas de replicação de elementos. Os cortes verticais indicam os três grupos de elementos com diferentes e independentes níveis de replicação, propriedades de segurança e dependabilidade. Cada um dos elementos pode assumir um modelo de faltas distinto e prover garantias diferenciadas em termos de funcionalidades oferecidas (e.g. garantia de entrega das mensagens).

Como o objetivo é garantir que o cliente consiga acessar o serviço alvo, este pode estar replicado de forma a garantir a disponibilidade do serviço. Pode-se assumir que a replicação do serviço alvo e do *gateway* é simples, ou seja, apenas para lidar com falhas por paragem. Não obstante, esses elementos podem também utilizar técnicas mais sofisticadas como tolerância a faltas arbitrárias, que demandam algoritmos e mecanismos mais sofisticados e complexos e têm um custo mais elevado. Entretanto, o modelo funcional proposto assume que faltas Bizantinas são toleradas apenas nos pontos mais críticos da arquitetura.

Tanto o serviço alvo quanto o *gateway* podem tolerar pelo menos falhas por pa-

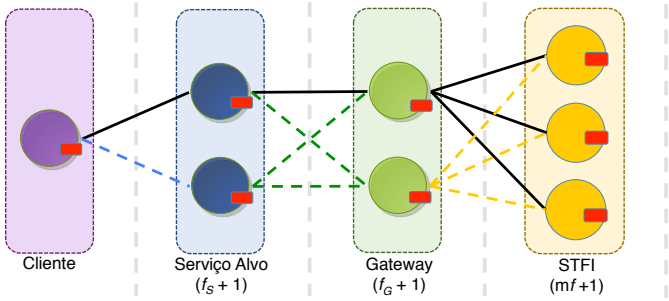

Figura 2. Modelo funcional estendido

ragem (f_S e f_G, respectivamente) no processo de acesso e utilização dos recursos do STFI. Complementarmente, ambos os elementos podem suportar detecção de alguns tipos de faltas arbitrárias, como violação de integridade ou autenticidade de mensagens. A verificação das mensagens pode ser realizada através da utilização, ou não, dos componentes seguros. Caso alguma informação criptográfica sigilosa seja necessária na verificação, componentes seguros devem ser utilizados. Caso contrário, métodos simples de verificação de pacotes podem ser aplicados.

Os clientes podem conectar-se a qualquer uma das réplicas do serviço alvo (e.g. serviço de autenticação de um site de comércio eletrônico). As réplicas deste podem conectar-se a quaisquer réplicas do *gateway*. O acesso às réplicas dos serviços ou *gateway*s pode ser baseado em informação contida em listas simples, como ocorre na prática em protocolos do tipo AAA, ou *round-robin* para balanceamento de carga, como é oferecido no serviço de DNS. No modelo proposto não há necessidade estrita de balanceamento de carga, uma vez que o maior objetivo é prover tolerância a faltas e intrusões. Sendo assim, em princípio, uma lista de endereços das réplicas é o suficiente, seja ela provida por algum serviço externo, como DNS ou algum tipo de diretório de descoberta, ou através de configuração direta dos elementos da arquitetura. Um serviço alvo, por exemplo, vai tentar a próxima réplica do *gateway* toda vez que não receber uma resposta ou receber uma mensagem corrompida da réplica atual.

O STFI foge à excessão da lista simples e método variável de acesso às réplicas. O *gateway* precisa conhecer todas as réplicas necessárias para suportar o limite de faltas estabelecido no sistema. Tomando como exemplo uma solução de $2f + 1$ réplicas no STFI, o *gateway* deve conhecer pelo menos $2f+1$ réplicas de forma a assegurar que faltas arbitrárias no STFI serão mascaradas enquanto que $f + 1$ réplicas estiverem funcionando corretamente.

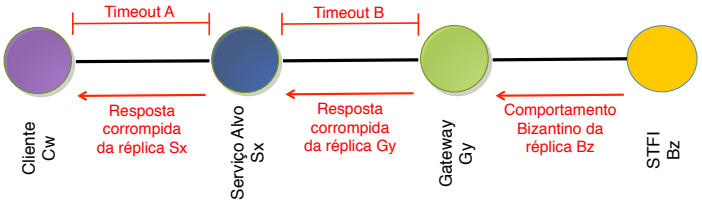

Figura 3. Mecanismos para detectar réplicas faltosas

A Figura 3 ilustra os mecanismos de detecção de faltas entre os componentes do modelo funcional. As alternativas de detecção entre os clientes, serviços alvo e *gateway*s

são baseadas em *timeouts* e mensagens corrompidas. Já entre os *gateway*s e STFIs são empregados mecanismos mais fortes, como protocolos de replicação de máquina de estados, a fim de tornar possível o mascaramento de faltas arbitrárias. Isso significa que nenhum comportamento anormal (e.g. atrasos, respostas mal-formadas, mensagens sintaticamente corretas porém maliciosas) de até f réplicas B_z irá afetar ou atrasar a operação do sistema. Mensagens de réplicas comprometidas podem ser simplesmente descartadas sem afetar o sistema. É assumido que a maioria das réplicas (todas exceto f) são corretas e estão operando conforme esperado.

É assumido, também, que protocolos de tolerância a faltas Bizantinas são utilizados no STFI. O *gateway* recebe a resposta de todas as réplicas do STFI e decide qual é a resposta correta que deve ser encaminhada para o serviço alvo. Para alcançar este objetivo o STFI necessita de $mf + 1$ réplicas, onde m varia de acordo com o protocolo em particular (e.g. $m = 2$, $m = 3$). O caso normal é $m = 3$, levando a um sistema com $3f + 1$ réplicas para tolerar f faltas. Entretanto, existem alternativas para reduzir o número de réplicas do sistema para tolerar f faltas. Há soluções que utilizam Trusted Timely Computing Bases (TTCBs) [Correia et al. 2004] para reduzir o número de réplicas a $2f+1$. Além disso, outros trabalhos classificam as réplicas em ativas e inativas. Sendo assim, considerando o caso normal, o número de réplicas ativas pode ser reduzido a $1f+1$ [Distler et al. 2011]. As réplicas inativas ficam em estado de espera, prontas para entrarem em operação. Tão logo forem necessárias, são ativadas para resolver problemas de consenso.

A Tabela 1 resume o modelo e limites de faltas dos principais componentes da arquitetura. Ela também lista exemplos de componentes em cenários reais, como infraestruturas AAA e OpenID. Como pode ser observado, o serviço alvo e o *gateway* suportam tanto faltas por paragem quanto um sub-conjunto de faltas arbitrárias, conforme comentado anteriormente. O STFI é capaz de tolerar intrusões e ataques, uma vez que estes podem ser tratados como faltas Bizantinas, em até f_B réplicas simultaneamente. Por fim, o elemento seguro tolera apenas faltas por paragem. Para tolerar f_C faltas seriam necessários f_C+1 componentes seguros. Entretanto, por razões de simplicidade, é assumido que há apenas um componente seguro por elemento funcional do sistema.

Tabela 1. Modelo de faltas, limite de faltas e exemplo de componentes

Componente	Modelo de falta	No. de réplicas	Limite de faltas	Exemplo de componentes em cenários reais
Cliente	—	—	—	Usuário final, dispositivo de rede
Serviço alvo	Crash/ Bizantina	$f_S + 1$	f_S	Roteador WiFi, serviço de migração de rede utilizando OpenID, switch OpenFlow
Gateway	Crash/ Bizantina	$f_G + 1$	f_G	Novo componente compatível com o serviço alvo e o STFI
STFI	Bizantina	$mf_B + 1$	f_B	Servidor AAA RADIUS, servidor OpenID, NIB de um controlador OpenFlow
Comp. seguro	Crash	1	0	Chaves TLS e funções de criptografia

3.3. Desenvolvendo Sistemas Resilientes e Confiáveis

Blocos de Construção

Os principais blocos de construção para implementação de arquiteturas de serviço no contexto do modelo proposto podem ser caracterizados em cinco.

Máquinas virtuais. Características como simplicidade de migração, manutenção e gerenciamento tornam as máquinas virtuais e tecnologias de virtualização atrativas para

a implantação e gestão de infraestruturas de serviços de rede.

Base de computação confiável é um bloco de construção quase que natural com a utilização de *hypervisors*. Pode-se assumir que um *hypervisor* forma uma base de computação confiável. Apesar disso, não é necessário confiar em todo o *hypervisor*. Alternativas como microkernels seguros [Heiser et al. 2007], capacidades de autoproteção [Wang and Jiang 2010] e operações locais confiáveis, como inicialização e paragem de máquinas virtuais, podem ser asseguradas de forma confiável.

Componentes seguros são pequenos e confiáveis componentes de software e/ou hardware capazes de assegurar propriedades ou funcionalidades críticas em um sistema, como confidencialidade. Os componentes seguros podem ser utilizados em diferentes partes da arquitetura. Em uma solução de autenticação baseada em OpenID, tanto o usuário quanto o servidor OpenID podem confiar os seus dados sensíveis e funções a um componente confiável. Assim, mesmo que um servidor seja comprometido não haverá vazamento de informações críticas, como chaves secretas.

Protocolos de replicação representam um dos principais componentes de sistemas resilientes. Replicação de máquina de estados e sistemas de quórum representam as duas abordagens mais comuns para o tratamento de faltas por paragem e arbitrárias [Verissimo et al. 2009]. Réplicas permitem ao sistema tolerar até f faltas simultâneas sem comprometer a operação do serviço. Esses protocolos, quando combinados com outras técnicas como diversidade e recuperação reativa-proativa, representam uma solução robusta no projeto e desenvolvimento de serviços resilientes.

Comunicação segura fim-a-fim é necessária para alcançar confidencialidade e privacidade dos dados do cliente. Protocolos como TLS e TTLS podem ser utilizados para prover canais confiáveis, autenticação mútua (e.g. EAP-TLS) e verificação de autenticidade do servidor.

Sistemas Distribuídos Híbridos

Serviços de rede resilientes podem ser projetados empregando conceitos de sistemas distribuídos híbridos. Estes combinam sistemas distribuídos homogêneos com componentes menores, confiáveis e previsíveis. Esses pequenos e especializados componentes permitem construir sistemas com garantias temporais e propriedades fortes de segurança. Não obstante, técnicas como replicação de máquina de estados são necessárias para garantir propriedades como integridade e disponibilidade do sistema.

O modelo de sistemas distribuídos híbridos [Veríssimo 2006], denominado de *wormhole*, propõe a segmentação do sistema em dois domínios, um com pressupostos fracos (e.g. assíncrono) e outro com pressupostos mais fortes (e.g. parcialmente síncrono). Enquanto que o domínio com os pressupostos fracos representa a maior parte e a maioria dos componentes e funcionalidades do sistema, o segundo domínio é formado de componentes e funcionalidades bastante específicas, requerendo propriedades e garantias como às fornecidas pelos componentes seguros. O domínio com pressupostos mais fortes pode ainda ser formado de um sub-sistema local ou distribuído. Em síntese, o conceito do *wormhole* propõe um sub-sistema com propriedades privilegiadas que pode ser utilizado para assegurar operações críticas de uma maneira segura e previsível.

Protocolos de Replicação

Os protocolos de replicação são peças-chave de uma arquitetura para serviços de rede resilientes. Existem diferentes tipos de protocolos, com diferentes proprieda-

des e garantias. BFT-SMaRt [Sousa and Bessani 2012] e IT-VM [Lau et al. 2012] são dois exemplos de protocolos recentes e em desenvolvimento. Ambos exploram técnicas e otimizações de protocolos anteriores, apresentando características e ganhos particulares em seus respectivos contextos. Enquanto o primeiro explora cenários de sistema que requerem alta disponibilidade, o segundo visa melhorar o desempenho de sistemas tolerantes a intrusões que utilizam máquinas virtuais sobre uma única máquina física.

BFT-SMaRt é uma biblioteca que oferece um serviço de BFT utilizando replicação de máquinas de estado. A arquitetura de uma réplica é composta por três grandes blocos, o Mod-SMaRt, o VP-Consensus e canais de comunicação confiáveis e autenticados [Sousa and Bessani 2012]. Mod-SMaRt é um algoritmo de replicação de máquina de estados otimizado para latência e resiliência. O VP-Consensus representa uma adaptação de algoritmos de consenso bem conhecidos e comprovados, como o Paxos. O principal diferencial está no fato de o VP-Consensus evitar protocolos de *broadcast* confiável e reduzir o número de passos de comunicação, quando comparada com soluções similares.

IT-VM é um algoritmo projetado para desenvolver sistemas tolerantes a intrusões onde as réplicas executam em máquinas virtuais e utilizam memória compartilhada na execução do protocolo de replicação de máquina de estados. Isso reduz a necessidade de comunicação através da rede, como é o caso das soluções que consideram o uso de múltiplas máquinas físicas. Portanto, o principal benefício é o maior desempenho, em detrimento de uma menor disponibilidade (e.g. dependência de uma única máquina física) e maior suscetibilidade a ataques de exaustão de recursos em ambiente virtualizados.

Componentes Seguros

Um componente seguro pode ser definido como um elemento pequeno e confiável com (a) um número de interfaces e quantidade de código reduzidas; (b) meios de ser fácil de garantir a corretude e correção de erros; (c) número de acessos ao dispositivo limitado, em alguns casos. As razões para a limitação de acessos podem incluir o fato do componente seguro ser um dispositivo lento quando comparado com o restante do sistema, de uma unidade comprometida do sistema poder causar sobrecarga de operação no componente seguro, ou ainda para evitar ataques de DoS.

Tabela 2. Interface do componnete TLS seguro

Método	Entrada	Saída
`generateRandom`	`Tamanho` (em bytes) do número aleatório.	`Número aleatório` com o tamanho requisitado.
`extractPreMaster`	`Segredo premaster` enviado pelo cliente.	`Verdadeiro` se o *premaster* foi decifrado corretamente, `falso` caso contrário.
`generateMaster`	Números aleatórios do cliente e do servidor.	`Verdadeiro` se o segredo *master* foi gerado, `falso` caso contrário.
`getServerFinishMsg`	`Hash` atual do *record stream*.	Mensagem de finalização do servidor.

A Tabela 2 mostra um exemplo de métodos necessários para um componente TLS seguro. Este componente foi implementado como componente de software isolado e utilizado na implementação dos protótipos. Para assegurar autenticação mútua fim-a-fim em serviços de rede como o RADIUS e o OpenID, componentes seguros podem ser utilizados em ambos os lados, no usuário final e no serviço de autenticação, de forma similar ao que foi proposto em [Urien et al. 2010], por exemplo.

Um componente TLS seguro precisa ser projetado para prover os mecanismos criptográficos necessários para a execução de um *handshake* TLS. Como detalhado na

tabela, há quatro métodos públicos. Qualquer software externo pode invocar os métodos públicos para executar um *handshake* entre o cliente e o serviço de autenticação. Entretanto, nenhum dado crítico deixa o componente seguro. A chave privada do servidor é necessária para decifrar o *premaster* (enviado pelo cliente utilizando o certificado público do servidor) e para gerar a *master key*. Sendo assim, um atacante não é capaz de comprometer a confidencialidade do sevidor de autenticação. Adicionalmente, usuário e servidor podem realizar autenticação mútua e, consequentemente, estabelecer uma relação de confiança entre ambos.

3.4. Propriedades e Requisitos do Serviço

A Tabela 3 expressa as propriedades e requisitos/componentes para o projeto e implantação de serviços com diferentes níveis de resiliência e confiabilidade. A maioria dos serviços disponíveis e utilizados para identificação e autenticação de usuários em organizações pertence às classificações 1 a 3 ($--$ = só *primary-backup*), ou seja, serviços geralmente não confiáveis (do ponto de vista de segurança) e pouco resilientes.

Tabela 3. Propriedades e requisitos/componentes do serviço

	Propriedades	Compontes seguros	Protocolos replicação	Modelo wormhole	Tolerância a Intrusões
1.	Não confiável	não	não	não	não
2.	Confiável e não resiliente	sim	não	não	não
3.	Resiliente ($--$) e não confiável	não	sim	não	não
4.	Resiliente e não confiável	não	sim	não	não
5.	Resiliente e não confiável	não	sim	sim	sim/não
6.	Resiliente e confiável	sim	sim	sim	sim
7.	Resiliente e confiável ($++$)	sim	sim	sim	sim

Por outro lado, o modelo funcional proposto visa contribuir para o desenvolvimento de serviços de rede com as propriedades 4 a 7 ($++$ = múltiplos pontos de verificação). Nas propriedades 4 e 5 pode ser observado que a primeira não utiliza o modelo de *wormhole*. Isso significa que um sistema com essa propriedade não pode ser assíncrono, uma vez que não é possível garantir que algoritmos de consenso irão finalizar a execução. Para garantir a finalização dos algoritmos de consenso, protocolos básicos dos sistemas replicados, é necessário prover garantias temporais mínimas. Um *wormhole* é capaz de garantir as propriedades temporais mínimas para a parte assíncrona do sistema.

Não obstante, a propriedade 5 pode levar a um sistema tolerante a intrusões, ou não. Caso o serviço não armazene nenhum dado sensível, ele pode ser tolerante a intrusões. Entretanto, se o servidor contém dados críticos que podem comprometer a confiabilidade do sistema ou viabilizar ações maliciosas como roubo de dados, o serviço não será tolerante a intrusões pelo fato de não garantir propriedades como confidencialidade.

As propriedades 6 e 7 requerem um *wormhole*, seja para garantir requisitos temporais, requisitos de segurança, ou ambos. Um *wormhole* utiliza componentes seguros para garantir as propriedades mínimas e funcionalidades críticas mínimas do sistema.

3.5. Cenários de Aplicação

A Figura 4 apresenta os *trade-offs* das diferentes configurações dos serviços em operação. Embora existam ganhos de desempenho quando são utilizadas soluções como o IT-VM, esta mesma solução pode sofrer impactos negativos de desempenho quando há ataques de exaustão de recursos. Mesmo assim, dependendo das necessidades e requisitos do

ambiente alvo, o IT-VM pode ser a solução, em termos de protocolo de replicação de máquina de estados, mais adequada. Em contrapartida, uma solução utilizando técnicas como as providas pelo BFT-SMaRt permite criar serviços mais robustos e com maiores índices/previsões de disponibilidade, uma vez que podem explorar os recursos e mecanismos de defesa de múltiplas máquinas físicas e/ou múltiplos domínios. Entretanto, a custo de um menor desempenho. Portanto, fica evidente que não existe uma única solução para todos os problemas. Os mecanismos e protocolos a serem empregados devem ser analisados sob a perspectiva dos requisitos e garantias necessárias ao ambiente alvo.

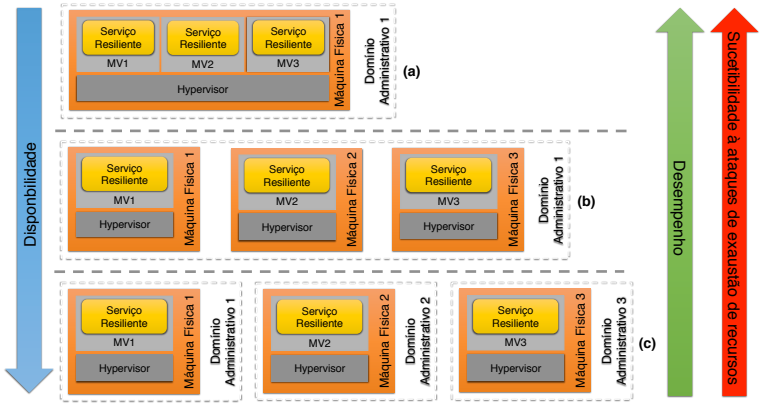

Figura 4. Configurações de implantação do serviço.

Um serviço resiliente que utiliza máquinas distribuídas em diferentes domínios (e.g. diferentes *clouds*) é capaz de tolerar tanto incidentes físicos em máquinas ou domínios (problemas de conexão de rede, faltas de energia, entre outros) quanto lógicos (má configuração de sistema ou rede, ataques de exaustão de recursos) apoiando-se também no suporte dos mecanismos e recursos oferecidos pelos diferentes domínios. Em termos práticos, já foi demonstrado que provedores *cloud* podem ser capazes de tolerar ataques de DDoS de grandes proporções sem prejuízos aos clientes [Prince 2012, Prince 2013]. Um dos recursos contra esse tipo de ataque são os múltiplos datacenters espalhados geograficamente, formando uma infraestrutura diversificada e robusta.

4. Implementações e Resultados

4.1. Protótipos

Para avaliação do modelo proposto foram desenvolvidos e avaliados dois protótipos tolerantes a faltas e intrusões, um do serviço OpenID e outro de um servidor RADIUS. Ambos os protótipos, OpenID e RADIUS [Malichevskyy et al. 2012], utilizaram o BFT-SMaRt. Isto é, ambas as soluções oferecem alta disponibilidade através da robustez e recursos de múltiplas máquinas físicas e/ou domínios administrativos.

É importante ressaltar que os protótipos do OpenID resiliente e do RADIUS resiliente mantêm compatibilidade com os padrões existentes. Em outras palavras, um serviço Web ou uma aplicação projetada para funcionar com um serviço OpenID tradicional vai funcionar da mesma forma com a versão tolerante a faltas e intrusões. A seguir é apresentada uma descrição sucinta do OpenID BFT, uma vez que o outro protótipo está detalhado na publicação a pouco citada.

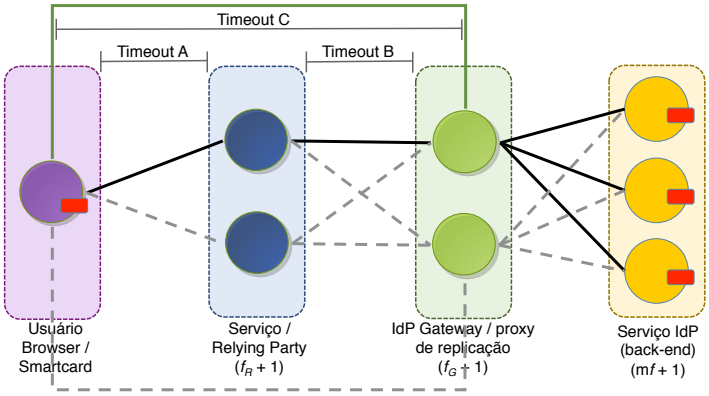

Figura 5. Modelo funcional do OpenID BFT-SMaRt

OpenID BFT-SMaRt

A arquitetura do OpenID replicado, com base no modelo funcional proposto, é ilustrada na Figura 5. As linhas tracejadas indicam alternativas no caso de alguma falha nas réplicas do caminho padrão (linhas sólidas). Nessa arquitetura, um cliente acessa um serviço Web (*relying party*) que redireciona o cliente para seu respectivo provedor OpenID, representado pelo *gateway* de replicação (IdP Gateway). Sendo que a identificação do usuário é processada pelas réplicas do Serviço IdP.

O TLS é adicionado como obrigatório para uma autenticação fim-a-fim através de um canal protegido, evitando problemas como os derivados de ataques que podem provocar o vazamento de informações através do *Referer* e redirecionamentos de URLs [Uruena et al. 2012]. O protótipo utiliza a biblioteca OpenID4Java [OpenID4Java 2013] (versão 0.9.8), a qual oferece suporte às versões 1.0 e 2.0 de autenticação do OpenID, sendo a última definida como padrão para a implementação.

Como o cliente e o servidor OpenID compartilham chaves com o objetivo de verificar a validade das mensagens na etapa de autenticação, foi necessária uma solução similar a apresentada para o RADIUS replicado [Malichevskyy et al. 2012], ou seja, uma forma de fazer com que as réplicas deterministas gerem a mesma chave para o cliente. E isso sem comprometer a chave, como detalhado em [Malichevskyy et al. 2012]. Apenas réplicas corretas, com o seu estado atualizado, conseguem reproduzir as mesmas chaves. A geração da chave, além de determinista entre réplicas corretas, é realizada dentro do componente seguro, ou seja, fora do alcance de um possível invasor.

4.2. Resultados e Discussão

A Tabela 4 apresenta o número de faltas de cada elemento referente ao seu respectivo protótipo. Como pode ser observado, as configurações são iguais. Ambas as soluções utilizam o BFT-SMaRt e, portanto, neste caso requerem $3f + 1$ (no *back-end*) réplicas para tolerar até f faltas ou intrusões simultâneas. Já numa implementação que faça uso de algoritmos semelhantes ao IT-VM, como o suposto Serviço SM, o número de réplicas do serviço crítico (no caso o *back-end*) reduz para $2f+1$, sendo que apenas $f+1$ réplicas precisam estar ativas para os casos de execuções sem faltas. Quando ocorre uma falta, as f réplicas adicionais podem ser ativadas para resolver o consenso.

Na Tabela 5 são expressadas as principais propriedades e características dos protótipos implementados. Dados sobre vazão, latência e tolerância a faltas e intrusões,

Tabela 4. Modelo funcional aplicado aos protótipos

Elemento	OpenID BFT	RADIUS BFT	Serviço SM
Serviço alvo	$1f + 1$	$1f + 1$	$1f + 1$
Gateway	$1f + 1$	$1f + 1$	$1f + 1$
Back-end	$3f + 1$	$3f + 1$	$2f + 1$

do protótipo RADIUS BFT, podem ser vistos em [Malichevskyy et al. 2012]. Como pode ser observado na tabela, as propriedades e características do OpenID BFT e RADIUS BFT são similares, pois ambas utilizam a mesma organização de componentes do modelo funcional e usufruem da biblioteca BFT-SMaRt e componentes seguros para alta disponibilidade e tolerância a faltas e intrusões. Além disso, é apresentando o Serviço SM, cujo objetivo é servir apenas de parâmetro para análise e comparação. É assumido que este serviço seja implementado com uma solução que explore memória compartilhada entre máquina virtuais, como é o caso do algoritmo IT-VM.

Tabela 5. Propriedades e características dos protótipos

Propriedade/suporte	OpenID BFT	RADIUS BFT	Serviço SM
1. Múltiplas máquinas físicas	sim	sim	não
2. Componentes confiáveis	sim	sim	sim
3. Hypervisor deve ser confiável e seguro	não	não	sim
4. Suscetibilidade a ataques de exaustão	baixa	baixa	média/alta
5. Desempenho (operações por segundo)	moderado	moderado	alto
6. Garantias de disponibilidade	alta	alta	moderada
7. Suporte a falhas Bizantinas	sim	sim	sim
8. Tolerância a intrusões	sim	sim	sim

A suscetibilidade a ataques de depleção está relacionada com o uso de máquinas virtuais em um mesmo *hypervisor*. Exemplos de ataques de exaustão de recursos e detalhes sobre o impacto no *hypervisor* Xen podem ser vistos em [Kreutz et al. 2013]. A suscetibilidade aos ataques é média a alta, uma vez que há ataques que podem comprometer em até mais de 50% o desempenho de máquinas virtuais não-maliciosas, quando as máquinas virtuais do serviço estão todas a operar sobre uma única máquina física e o mesmo *hypervisor*.

O desempenho de uma solução que utiliza máquinas virtuais em uma mesma plataforma de hardware, utilizando memória compartilhada para os protocolos de comunicação, como o consensus, é o melhor caso em termos de desempenho para sistemas tolerantes a faltas e intrusões. Por isso que o seu desempenho é considerado como alto. Já no caso de soluções que necessitem de troca de mensagens, o desempenho depende dos algoritmos utilizados e da respectiva implementação. No caso do OpenID BFT e RADIUS BFT, como foi utilizado o BFT-SMaRt, que implementa conjuntos de otimizações para replicação de máquina de estados [Sousa and Bessani 2012], o desempenho pode ser considerado como moderado. Já outras soluções, utilizando protocolos não otimizados, levariam o sistema a um desempenho inferior, podendo ser classificadas como de baixo desempenho. Em comparação com sistemas não tolerantes a faltas e intrusões, há naturalmente uma diferença de latência, que pode chegar a uma ordem de grandeza, e vazão, como número de autenticações por segundo, cuja diferença pode variar dependendo da configuração do sistema e dos recursos utilizados (como pool de threads), como pode ser visto em [Malichevskyy et al. 2012]. Entretanto, observa-se que

a diferença na vazão é menor que na latência.

As decisões de desenvolvimento das soluções dependem do nível de segurança e dependabilidade desejados. Uma solução como o Serviço SM é mais apropriada para ambientes onde a probabilidade de ataques de exaustão de recursos é baixa e o índice de disponibilidade do serviço é aceitável quando comparado com sistemas não resilientes. Por outro lado, soluções como OpenID BFT e RADIUS BFT são mais robustas e indicadas para ambientes de alta disponibilidade, onde o desempenho não é o maior ou principal objetivo, ou onde o *hypervisor* não é confiável, ou ainda quando ataques dos mais diversos tipos, como exaustão de recursos, podem ocorrer. Estas soluções são capazes de resistir a diferentes tipos de ameaças ou incidentes na infraestrutura, uma vez que as réplicas podem estar espalhadas em diferentes domínios administrativos e máquinas físicas.

5. Conclusão

De forma a contribuir para o desenvolvimento de soluções mais robustas e confiáveis de serviços de rede, este trabalho apresentou o primeiro modelo funcional para o projeto e desenvolvimento de serviços de identificação e autenticação, como OpenID e RADIUS, tolerantes a faltas e intrusões. Este é um importante passo para o desenvolvimento de contra-medidas às ameaças que circulam a grande rede.

O modelo foi analisado e discutido com base em dois protótipos distintos. Os resultados demonstram que a proposta é aplicável a diferentes contextos e cenários, o que pode envolver propriedades e requisitos como desempenho, alta disponbilidade, confiabilidade e confidencialidade. Além disso, vale ressaltar que o modelo proposto não restringe-se aos casos apresentados, podendo ser utilizado em quaisquer outros serviços com características similares.

O principal desafio da definição do modelo, bem como da implementação dos protótipos, está relacionado a compreender os diferentes elementos, blocos de construção e tecnologias disponíveis para dar o suporte necessário, tornando possível garantir as propriedades desejadas de cada cenário em particular. Na parte prática, envolve o fato de conhecer e conceber sistemas sob a ótica e perspectiva de segurança e dependabilidade, como protocolos de replicação de máquinas de estado e componentes seguros. Definir onde e quando são necessários é um exercício de engenharia importante.

Agradecimentos

Agradecemos aos revisores anônimos pelos comentários. Este trabalho é suportado pela FCT, através do Programa Multianual (LaSIGE) e projeto TRONE (CMU-PT/RNQ/0015/2009), pela Comissão Européia, através do projeto SecFuNet (FP7-ICT-STREP-288349), e pelo CNPq, através dos processos de número 590047/2011-6 e 202104/2012-5.

6. Referências

Alchieri, E. A. P., Bessani, A. N., and Fraga, J. d. S. (2008). A dependable infrastructure for cooperative web services coordination. In *IEEE ICWS*.

Clamshell (2013). Clamshell: An OpenID Server. `goo.gl/09pYF`.

Correia, M., Neves, N., and Verissimo, P. (2004). How to tolerate half less one byzantine nodes in practical distributed systems. In *IEEE SRDS*.

Distler, T., Kapitza, R., Popov, I., Reiser, H. P., and Schröder-Preikschat, W. (2011). Spare: Replicas on hold. In *NDSS*.

Feng, J. (2009). Analysis, Implementation and Extensions of RADIUS Protocol. In *International Conference on Networking and Digital Society*.

FreeRADIUS (2012). Documentation. goo.gl/6g8Qy.

Heiser, G., Elphinstone, K., Kuz, I., Klein, G., and Petters, S. M. (2007). Towards trustworthy computing systems: taking microkernels to the next level. *SIGOPS Oper. Syst. Rev.*, 41(4):3–11.

Juniper Networks (2010). Steel belted radius carrier 7.0 administration and configuration guide. goo.gl/Y5b9k.

Kreutz, D., Niedermayer, H., Feitosa, E., da Silva Fraga, J., and Malichevskyy, O. (2013). Architecture components for resilient networks. http://goo.gl/xBHCNb.

Lau, J., Barreto, L., and da Silva Fraga, J. (2012). An infrastructure based in virtualization for intrusion tolerant services. In *IEEE ICWS*.

Leicher, A., Schmidt, A., Shah, Y., and Cha, I. (2010). Trusted computing enhanced openid. In *ICITST*, pages 1–8.

Malichevskyy, O., Kreutz, D., Pasin, M., and Bessani, A. (2012). O vigia dos vigias: um serviço RADIUS resiliente. In *INForum*.

OpenID (2010). OpenID community wiki. goo.gl/PCASy.

OpenID4Java (2013). Openid 2.0 java libraries. goo.gl/c3kFV.

Prince, M. (2012). Ceasefires Don't End Cyberwars. goo.gl/GI506.

Prince, M. (2013). The DDoS That Almost Broke the Internet. goo.gl/g5Qs1.

RADIUS Partnerships (2008). Deploying RADIUS: Practices and Principles for AAA solutions. goo.gl/fslu7.

Recordon, D. and Reed, D. (2006). OpenID 2.0: a platform for user-centric identity management. In *2nd ACM workshop on Digital identity management*. ACM.

Rigney, C., Willens, S., Rubens, A., and Simpson, W. (2000). RFC 2865 - Remote Authentication Dial In User Service (RADIUS).

Sousa, J. and Bessani, A. (2012). From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation. In *Ninth EDCC*.

Sun, S.-T., Hawkey, K., and Beznosov, K. (2012). Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. *Computers & Security*, 31(4):465–483.

Tankard, C. (2011). Advanced persistent threats and how to monitor and deter them. *Network Security*, 2011(8).

Urien, P., Marie, E., and Kiennert, C. (2010). An innovative solution for cloud computing authentication: Grids of EAP-TLS smart cards. In *Fifth ICDT*.

Uruena, M., Munoz, A., and Larrabeiti, D. (2012). Analysis of privacy vulnerabilities in single sign-on mechanisms for multimedia websites. *Multimedia Tools and Apps*.

van Delft, B. and Oostdijk, M. (2010). A Security Analysis of OpenID. In *IFIP Advances in Information and Comm. Tech.*, volume 343. Springer.

Verissimo, P., Correia, M., Neves, N. F., and Sousa, P. (2009). Intrusion-resilient middleware design and validation. *Information Assurance, Security and Privacy Services*, 4:615–678.

Veríssimo, P. E. (2006). Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1):66–81.

Verizon RISK Team (2013). Data breach investigations report. Technical report. goo.gl/7mIBy.

Wang, Z. and Jiang, X. (2010). Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *IEEE SSP*, pages 380–395.

