SBSeg 2005

V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Florianópolis, 26 a 30 de Setembro de 2005

Livro Texto dos Minicursos

SBSeg 2005

V SIMPÓSIO BRASILEIRO EM SEGURANÇA
DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS

Florianópolis, 26 a 30 de setembro de 2005

Livro Texto dos Minicursos

EDITORES
Luciano Paschoal Gaspary
Frank Siqueira

ORGANIZAÇÃO
Universidade Federal de Santa Catarina

PROMOÇÃO
Sociedade Brasileira de Computação

S617l Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (5. : 2005 : Florianópolis, SC)
Livro texto dos minicursos / V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais ; editores: Luciano Paschoal Gaspary e Frank Siqueira; organização Universidade Federal de Santa Catarina. – Florianópolis : Sociedade Brasileira de Computação, 2005.
168p.

ISBN:85-7669-047-0

1. Informática. 2. Segurança de sistemas computacionais. 2. Segurança da Informação. I. Gaspary, Luciano Paschoal. II. Siqueira, Frank. III. Título.

CDU: 681.3

Catalogação na publicação por: Onélia Silva Guimarães CRB-14/071

Prefácio

Os minicursos do SBSeg têm como objetivo (i) atender a uma necessidade de atualização em temas normalmente não cobertos nas grades curriculares ou (ii) despertar grande interesse entre acadêmicos e profissionais. Já na primeira edição da Chamada de Minicursos, a comunidade prestigia a mesma com um elevado número de submissões. De um total de dezessete propostas submetidas, o comitê de avaliação selecionou três para apresentação e publicação na forma de capítulo de livro. Recebemos propostas de autores vinculados a entidades públicas e privadas do País e do exterior.

O processo de avaliação das propostas contou com a participação direta dos sete membros do comitê de avaliação. Cada proposta foi revisada por, pelo menos, três especialistas na área.

Neste livro encontram-se os textos completos das propostas selecionadas, organizados em três capítulos: (1) Negação de Serviço: Ataques e Contramedidas; (2) Segurança em Grades Computacionais; (3) Serviços Distribuídos Tolerantes a Intrusões: Resultados Recentes e Problemas Abertos.

Agradeço penhoradamente o inestimável trabalho realizado pelos membros do comitê de avaliação durante o processo de seleção das propostas e aos autores pela alta qualidade de suas contribuições. Por fim, desejo manifestar minha gratidão à comissão organizadora do SBSeg 2005, em especial ao professor Frank Siqueira, por apoiar e tornar possível a realização dos minicursos.

Vida longa aos Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais!

São Leopoldo, agosto de 2005

Luciano Paschoal Gaspary
Coordenador de Minicursos do SBSeg 2005

Realização

Coordenação Geral
Frank Siqueira (UFSC)

Comitê de Avaliação de Minicursos

Coordenação
Luciano Paschoal Gaspary (UNISINOS)

Membros
Antonio Alfredo Ferreira Loureiro (UFMG)
Carlos Alberto Maziero (PUC-PR)
Frank Siqueira (UFSC)
Paulo Barreto (USP)
Ricardo Dahab (UNICAMP)
Thaís Vasconcelos Batista (UFRN)

Sumário

Capítulo 1

Negação de Serviço: Ataques e Contramedidas

Rafael P. Laufer[1], Igor M. Moraes[1], Pedro B. Velloso[1,2],
Marco D. D. Bicudo[1], Miguel Elias M. Campista[1], Daniel de O. Cunha[1],
Luís Henrique M. K. Costa[1] e Otto Carlos M. B. Duarte[1]

[1]Grupo de Teleinformática e Automação – GTA*
COPPE-Poli – Universidade Federal do Rio de Janeiro

[2]Laboratoire d'Informatique de Paris 6 – LIP6
Université Pierre et Marie Curie - Paris VI

Abstract

The Internet architecture exposes the users to different types of digital attacks and threats. Denial of service (DoS) is a common attack which may cause huge damage to a victim. The prevention of such attacks is a challenge even for overprovisioned and up-to-date computers. As a consequence, the best way to inhibit DoS attacks is to be ready to react to those attacks on the fly. One such reactive mechanism is IP traceback. Such mechanism is important, since packets with spoofed source addresses are employed to disguise the actual source of the attack. This chapter presents the most common techniques used to combat DoS attacks in the Internet. Different IP traceback systems proposed in the literature are also analyzed in detail. At last, a stateless single-packet IP traceback system is introduced.

Resumo

A arquitetura da Internet expõe os usuários a diversos tipos de ataques e pragas digitais. Ataques comuns que causam enormes prejuízos à vítima são os ataques de negação de serviço. Prevenir-se contra tais ataques é um desafio, mesmo para computadores regularmente atualizados e superdimensionados. Por isso, a melhor maneira de inibir ataques de negação de serviço é estar pronto para reagir a estes ataques. Uma forma de inibição é adotar sistemas de rastreamento de pacotes IP. Tais sistemas são fundamentais, uma vez que pacotes com endereços IP de origem forjados podem ser usados no ataque para ocultar a verdadeira origem do atacante. Neste capítulo, são apresentadas as técnicas usadas para inibir ataques de negação de serviço na Internet. São caracterizados ainda diversos sistemas de rastreamento de pacotes IP propostos na literatura. Por fim, é introduzido um novo sistema de rastreamento de pacotes IP.

*Apoiado pelos recursos da CAPES, CNPq, FAPERJ, FINEP, RNP, FUNTTEL e UOL

1.1. Introdução

A Internet é um espaço virtual de encontros e ao mesmo tempo um local de conflitos, devido à diversidade cultural dos seus usuários. Esse ambiente cria um campo fértil para ações maliciosas, que exploram falhas de segurança existentes nas aplicações e nos protocolos de comunicação da Internet. Vírus são enviados por e-mail, infectando computadores e se disseminando sem serem notados. *Spams* enchem as caixas de mensagens dos usuários da rede. Usuários mal-intencionados, que vão desde quadrilhas muito bem estruturadas a meros adolescentes em busca de auto-afirmação, tentam invadir computadores e redes para roubar informações confidenciais ou apenas mostrar seu potencial. Ataques de negação de serviço (*Denial of Service* – DoS) consomem os recursos de servidores e roteadores e impedem que usuários legítimos tenham acesso a um determinado serviço. Enfim, a lista de ataques cresce a cada dia.

Estatísticas apontam um crescimento mundial de 226% no volume de mensagens eletrônicas não solicitadas (*spams*) de abril para maio de 2005 [IBM Report, 2005]. Neste mesmo período, o aumento no volume de e-mails contendo pragas digitais aumentou de 33%. O montante de e-mails não solicitados já representa 68,7% de todos os e-mails que circulam na Internet e os que contêm pragas digitais representam 3,12%. No Brasil esse problema é bem grave, pois atualmente o país é apontado como o 5^o maior receptor de *spams* no mundo [Globo Online, 2005]. Apesar de todos os esforços para bloqueá-los, muitos ainda não são filtrados devido ao avanço de técnicas para burlar os programas anti-*spam*. Provedores acabam aumentando o custo de sua operação por disponibilizar involuntariamente recursos para armazenar e enviar *spams*. No que se refere a negação de serviço na Internet, um estudo recente estima em mais de 4000 o número de ataques em um período de uma semana [Moore et al., 2001]. Um resultado importante mostra ainda que grande parte dos ataques são concentrados em vítimas brasileiras. De acordo com as informações analisadas, o domínio `.br` é o quarto domínio mais atacado por inundações visando a negação de serviço, concentrando cerca de 5 a 7% dos ataques. Em toda a Internet, somente os domínios `.net`, `.com` e `.ro` foram mais atacados que o domínio brasileiro. Uma informação do relatório anual do CSI/FBI (*Computer Security Institute/Federal Bureau of Investigation*) [Gordon et al., 2005] sobre crimes na área de computação afirma ainda que os ataques de negação de serviço estão entre os incidentes de segurança que mais causam prejuízo às instituições americanas.

Até o momento, não há uma solução completa para a maioria dos problemas de segurança da Internet. Porém, a ocorrência e os efeitos de alguns destes ataques foram reduzidos, através da adoção de determinadas ferramentas. O uso de *firewalls* reduz o número de invasões a redes de computadores, uma vez que, com exceção dos pacotes legítimos, todo o tráfego de entrada é bloqueado. Os programas antivírus previnem a execução de vírus e vermes conhecidos no computador no qual estão rodando. Dessa forma, os antivírus impedem a infecção do computador e interrompem a disseminação de tais pragas pela rede. Da mesma forma, os programas anti-*spam* buscam reduzir o número de mensagens não solicitadas recebidas. Entretanto, tanto os antivírus quanto os anti-*spams* têm que ser constantemente atualizados, pois novas pragas e técnicas para enviar mensagens não solicitadas surgem a cada dia. Sistemas operacionais e aplicações verificam periodicamente a existência de atualizações de software e automaticamente se remendam, o que reduz o número de vulnerabilidades que podem ser exploradas em um

computador. No entanto, nenhuma das ferramentas existentes é eficaz contra ataques de negação de serviço. Apesar do empenho das áreas acadêmica e industrial na busca de soluções para evitar e remediar tais ataques, as ferramentas existentes são capazes de lidar apenas com ataques pouco refinados. Por isso, os ataques de negação de serviço são, em sua maioria, bem sucedidos. Atualmente, é fácil burlar as ferramentas de defesa e negar o serviço da vítima durante o tempo desejado pelo atacante, sem que nenhuma medida efetiva possa ser tomada.

Contra ataques de negação de serviço, podem ser tomadas medidas preventivas e reativas. Para prevenir-se de um ataque, uma possível vítima deve atualizar regularmente seus programas e possuir recursos suficientes para suportar um ataque sem prejudicar seus usuários legítimos. Mesmo assim, tais práticas não garantem que um computador estará protegido contra ataques de negação de serviço. Novas vulnerabilidades são descobertas a cada dia e, mesmo para computadores superdimensionados, não é possível garantir a imunidade da vítima quando o tráfego de ataque é gerado por diversos atacantes. Dessa forma, por ser difícil evitar ataques de negação de serviço, é necessário adotar medidas para reagir a tais ataques. Uma medida reativa é rastrear os pacotes de ataque. Como a maioria dos ataques de negação de serviço utiliza pacotes com endereços IP de origem forjados, o objetivo do rastreamento é determinar a verdadeira origem e a rota percorrida por um pacote ou um conjunto de pacotes IP. Ou seja, a partir de pacotes recebidos pela vítima, deseja-se encontrar quem realmente os enviou, de forma a aplicar punições e inibir futuros ataques.

O objetivo principal deste capítulo é apresentar os conceitos e as técnicas usadas para inibir ataques de negação de serviço na Internet. Inicialmente são abordados alguns dos principais ataques na Internet, como a disseminação de pragas digitais e *spams*. Os tipos de ataques de negação de serviço são analisados assim como as contramedidas. Em seguida, são analisados ainda diversos sistemas de rastreamento de pacotes propostos na literatura, mostrando suas características e ressaltando suas virtudes e problemas. Por fim, é apresentado um novo sistema de rastreamento de pacotes que não armazena estado na infra-estrutura da rede e que é capaz de identificar um atacante utilizando apenas um pacote empregado no ataque.

1.2. Ataques na Internet

Os ataques na Internet podem ser realizados de diversas formas e com propósitos variados. Muitos ataques visam obter informações confidenciais, se apoderar de recursos computacionais para execução de uma determinada tarefa, anunciar produtos comerciais ou inviabilizar serviços. Dependendo do tipo de ataque, o alvo principal podem ser os computadores dos usuários, os servidores ou a própria rede. Assim, de alguma forma, a operação normal da rede é prejudicada pela incapacidade de comunicação entre os computadores. Na maioria das vezes, as vítimas não têm conhecimento dos ataques e podem inclusive colaborar involuntariamente, por exemplo, ao entrarem em um sítio divulgado por um *spam* ou ao executarem um programa malicioso por engano. Nesta seção, são abordados alguns dos principais ataques da Internet, como a disseminação de pragas digitais, como os vírus e os cavalos de Tróia, e o envio de *spams*.

1.2.1. Pragas Digitais

Os ataques associados a códigos maliciosos custam bilhões de dólares a cada ano. A maioria das organizações considera que as ações corretivas para estes ataques demandam muito tempo e um alto custo operacional. Em alguns casos, a recuperação dos prejuízos causados por um ataque e a normalização das operações cotidianas podem levar vários dias. Os profissionais da área de segurança despendem enormes esforços para aprimorar os métodos de defesa dos computadores dentro da rede de uma organização. Porém, à medida que estas defesas se tornam mais eficientes, os atacantes se adaptam às novas condições buscando novos caminhos para explorar vulnerabilidades.

Os primeiros programadores de códigos maliciosos eram principalmente pessoas conhecidas como *nerds* de computadores. Os *nerds* são considerados pessoas socialmente alienadas que podem invadir sistemas computacionais por diversão, para protestar contra regras sociais ou pela simples vaidade de mostrar sua capacidade ao atacar sítios de empresas de alta tecnologia ou de governos. Atualmente, este cenário se transformou. Hoje, muitos códigos maliciosos visam benefícios financeiros, ao tentar controlar computadores alheios para divulgação de produtos ou serviços através de mensagens de correio eletrônico, outros visam o enriquecimento ilícito, ao capturar dados pessoais e senhas de usuários do sistema financeiro. As perspectivas futuras são ainda piores devido à eficiência dos ataques, à dificuldade de se chegar ao atacante e à falta de uma legislação específica para os crimes digitais. Uma verdadeira "guerra informacional" entre atacantes com objetivos ilícitos e profissionais de segurança de sistemas computacionais e de redes de computadores se anuncia. Os alvos dos ataques são quaisquer pontos críticos da infra-estrutura de comunicação e computação, seja o usuário doméstico ou uma grande organização.

Os ataques de códigos maliciosos podem ser mascarados por uma vasta gama de aplicações legítimas e transportados por um número cada vez maior de mecanismos. Em geral, estas pragas digitais impedem o funcionamento normal de um computador ou de uma rede e a sua execução é comumente realizada sem o consentimento do usuário. Entender como estas pragas digitais operam é de grande interesse para o desenvolvimento de estratégias defensivas, para a seleção de produtos de segurança e para o preparo dos usuários contra ameaças em potencial. Esta seção explica os diferentes tipos de ataques por programas maliciosos. Os tipos de ataques maliciosos que caracterizam as principais pragas digitais são: os vírus de e-mail ou de outro tipo, os cavalos de tróia ou outro tipo de porta dos fundos (*backdoors*), os vermes (*worms*), o *spyware*, o *adware* e o *stealware*.

De maneira geral, as pragas se replicam e se espalham de um computador para outro. Os vírus, os vermes (*worms*) e os cavalos de Tróia são pragas que são confundidas.

Um vírus é um trecho de código que se anexa a um programa ou arquivo, de forma a poder se espalhar e infectar outros sistemas quando é transportado. Os efeitos nocivos dos vírus são bastante variados, indo de ações irritantes como a exibição de uma imagem na tela, em momentos aleatórios, até estragos no hardware, software ou sistemas de arquivos. A maioria dos vírus são anexados a arquivos executáveis, o que significa que o vírus pode estar presente no computador mas não provocará efeitos maliciosos a menos que o programa seja executado. É importante notar que o vírus não se espalha sem a ajuda de uma intervenção humana, como a execução de um programa. As pessoas ajudam a espalhar o vírus, muitas vezes sem saber, através do compartilhamento de arquivos ou

envio de arquivos contaminados anexados a e-mails.

Muitos vírus não têm uma tarefa específica a realizar e apenas se replicam indefinidamente, congestionando o sistema de e-mail. Por outro lado, alguns vírus possuem conteúdo malicioso, realizando tarefas como apagar ou corromper dados de arquivos ou desabilitar programas de segurança. Existem vírus que modificam um trecho de código de outro programa para facilitar a sua propagação. Geralmente, os vírus apresentam a seguinte estrutura:

- um mecanismo de replicação, que permite que os vírus se espalhem;
- um disparador, que é um evento externo ou interno para indicar o início da replicação do vírus, da tarefa que este realiza ou de ambas;
- e uma tarefa, ou seja, a ação deve ser executada.

Estes três componentes podem ter diversas formas e comportamentos. Os mecanismos de replicação variam consideravelmente. Um vírus pode executar uma incontável quantidade de combinações de tarefas. No entanto, alguns tipos de vírus são mais comuns e merecem destaque.

O vírus de setor de inicialização, ou setor de *boot*, infecta o primeiro setor de um disco rígido ou de um disquete. O primeiro setor destes discos contém o registro de inicialização mestre (*Master Boot Record* - MBR), que realiza configurações iniciais antes do carregamento do sistema operacional. Assim, quando o computador é ligado, o vírus é iniciado, antes mesmo do sistema operacional, tomando controle sobre qualquer recurso necessário. Geralmente, esse tipo de vírus infecta todo disquete que é inserido no computador. Um vírus de setor de *boot* geralmente não é transportado pela rede, sendo, portanto, cada vez mais incomum atualmente devido ao desuso do disquete.

Os vírus removedores têm a tarefa de apagar arquivos com nomes específicos que são necessários à execução básica do sistema operacional ou são essenciais a um editor de texto ou gráfico, por exemplo. Os vírus infecciosos geralmente se anexam a arquivos executáveis com extensões `.com` e `.exe`. Assim, a cada execução do programa infectado, o vírus se espalha ainda mais pelos arquivos do computador. Os vírus embutidos em conteúdo (*content-embedded viruses*) são vírus-infecciosos que residem, ou estão anexados, a arquivos de imagem, páginas em HTML (*HyperText Markup Language*), arquivos de vídeo ou arquivos de som. Os vírus de macro são semelhantes aos vírus embutidos em conteúdo, pois estão anexados a arquivos de editores como o Microsoft Word ou planilhas do Microsoft Excel. Como estas macros estão geralmente anexadas aos documentos e são executadas pelos programas, esses vírus podem utilizar disquetes, e-mail ou até um servidor de arquivos como meio de difusão.

Os vírus de e-mail em massa (*mass mailers viruses*) executam procedimentos do programa cliente de e-mail para se auto-replicar e enviar, através de um e-mail, para todos os endereços armazenados na agenda do usuário. Existe uma diferença no nível de ameaça entre vírus de e-mail em massa e vírus de e-mail comum. Ambos usam o mesmo método de replicação e de transporte, porém o que envia e-mails em massa é considerado mais

perigoso devido a sua taxa de replicação, à sobrecarga do sistema de correio eletrônico e à inundação da caixa de entrada dos usuários.

Os vírus furtivos (*stealth viruses*) se escondem dos softwares de detecção de vírus. Estes vírus são projetados para burlar as heurísticas utilizadas para detecção de vírus desconhecidos pelos softwares antivírus.

Os vírus de engenharia social são outro tipo cada vez mais comum. Junto com os boatos (*hoaxes*), que são definidos como brincadeiras ou "pegadinhas", este tipo de praga virtual convida o usuário a baixar e executar códigos maliciosos ou entrar em páginas que contenham códigos maliciosos.

Além destes tipos de vírus, existem vírus de características múltiplas, que combinam as funcionalidades e os atributos descritos acima.

O cavalo de Tróia tem este nome porque, tal como na lenda mitológica, o "presente" recebido aparentemente não oferece nenhum perigo. Ou seja, à primeira vista o programa que contém um cavalo de Tróia parece útil, no entanto uma vez instalado ele causará estragos no computador. Os cavalos de Tróia são códigos maliciosos que executam comandos não-autorizados no computador-alvo. Normalmente, as pessoas que instalam um programa com cavalo de Tróia o fazem por pensarem estar instalando software de uma fonte legítima. Os efeitos dos cavalos de Tróia podem ir desde simples ações irritantes, como a modificação da área de trabalho ou a criação de ícones indesejados, até prejuízos mais sérios como o apagamento de arquivos e destruição de informações. Alguns cavalos de Tróia podem criar "portas dos fundos" (*backdoors*) para liberar o acesso ao computador para usuários maliciosos executarem comandos remotamente, possivelmente permitindo que informação confidencial seja obtida. Diferentemente de vírus e vermes, os cavalos de Tróia não se replicam através da infecção de outros arquivos nem se auto-replicam.

Um verme é similar a um vírus em sua estrutura e pode ser considerado uma subclasse do vírus. O verme se alastra de um computador para outro, mas ao contrário de um vírus, o verme tem a capacidade de se alastrar sem a ajuda da intervenção humana. O verme se aproveita de algum arquivo ou de procedimentos de transferência que existem no sistema do computador que permitem que ele se propague sem nenhuma ajuda, ou seja, de forma autônoma. Estes computadores geralmente apresentam falhas de segurança exploradas pelo código malicioso do verme, que utiliza uma rede local ou uma conexão com a Internet. Quando o verme encontra um computador vulnerável (por exemplo, através de uma porta dos fundos), se replica, transfere sua réplica para o computador remoto e continua sua busca por novos alvos. A grande ameaça do verme é sua replicação no computador e, como conseqüência, o esgotamento da memória ou da capacidade de processamento da máquina. Devido à capacidade de replicação do verme, em vez do computador transferir um único verme ele pode transferir centenas ou milhares de suas cópias. Um exemplo de verme é o que faz transferência de uma cópia para toda a lista de correios eletrônicos que existe no computador.

O termo software-espião (*spyware*) designa qualquer programa que coleta informações sobre um indivíduo ou organização sem seu consentimento prévio. O software-espião pode ser instalado em um computador por diversos meios. O espião pode ser

introduzido como parte de um vírus, verme, cavalo de Tróia ou até mesmo como parte de programas legítimos. Vale ressaltar que os termos *spyware*, *adware* e *stealware* são comumente utilizados para descrever o mesmo tipo de código malicioso ou tipos semelhantes desta praga digital. Vários países estão em vias de banir legalmente, ou pelo menos controlar, este tipo de abuso digital. Os softwares-espiões são usados geralmente para colher informações com o fim de gerar estatísticas sobre senhas, hábitos do usuário, sítios da Internet acessados por um grupo de usuários de características específicas, sistema operacional utilizado, aplicações instaladas nos computadores, etc. Também podem ser coletadas informações de nomes e sobrenomes dos usuários, números de cartões de crédito e outras informações pessoais. Estas informações são geralmente combinadas com outras bases de dados de usuários e consumidores potenciais, criando assim perfis de indivíduos. Estes perfis podem então ser usados para fins de *marketing* e pesquisa de mercado.

O termo *adware*, que vem do inglês *advertisement software*, designa qualquer software ou parte de software, não necessariamente malicioso, que exibe propagandas enquanto uma aplicação principal é executada. Estas aplicações exibem na tela do usuário janelas de navegadores (*pop-ups*) ou do ambiente gráfico do usuário com anúncios de organizações patrocinadoras do software. Os *adwares* são geralmente encontrados em programas *shareware*, que são aplicativos que possuem licença de utilização temporária, ou programas *freeware*, que são gratuitos. Ambos estão comumente disponíveis na Internet. Através da comercialização destes anúncios, pequenas empresas de software ou programadores autônomos podem financiar o desenvolvimento de um software. Porém, um *adware* não designa apenas a utilização de aplicações como meio de propaganda. Um *adware* também pode se servir dos chamados defeitos de *Web* (*Web bugs*) para coletar informações sobre usuários de computadores e compilar perfis de usuários, se assemelhando ao comportamento de um *spyware*. Por exemplo, existem sítios da Internet que requisitam, às vezes compulsoriamente, aos usuários que acessam suas páginas que aceitem *cookies*, que são pequenos arquivos armazenados para monitorar o acesso dos usuários. Alguns desses *cookies* são considerados defeitos de *Web*, pois podem ser explorados como fonte de informação sobre o usuário e suas ações.

O *stealware* é um programa "ladrão de audiência". Na Internet, muitas vezes o sítio de uma organização possui atalhos (*links*) para o sítio de patrocinadores da organização. A organização é remunerada pela empresa patrocinadora através da contagem dos cliques que os usuários realizaram no *link* do patrocinador. O *stealware* burla este mecanismo, de forma a contabilizar os cliques realizados para uma outra organização, diferente da organização legítima. A identificação da organização é geralmente realizada por informações armazenadas na estação do usuário, como *cookies*, ou através do próprio endereço da página *Web* acessada. Normalmente, o *stealware* é embutido em softwares que alguns sítios obrigam o usuário a instalar, como requisito para liberar o acesso do usuário ao sítio. O usuário, ao executar o software que permite acesso ao conteúdo restrito, executa também o código malicioso. Tudo o que o *stealware* tem a fazer então é modificar os *cookies* na máquina do usuário ou forjar o endereço da página acessada.

1.2.2. Mensagem Eletrônica Não Solicitada ou *Spam*

O *spam*[†], ou mensagem eletrônica não solicitada, é toda a mensagem enviada sem a autorização do destinatário. Dentre as mensagens eletrônicas que são utilizadas como *spams* estão o correio eletrônico (e-mails), os torpedos (*Short Message Service* – SMS ou *Multimedia Message Service* – MMS) e as mensagens instantâneas (*Instant Messenger*). O indivíduo que gera e envia *spams* é chamado de *spammer*.

A definição de *spam* pelo Grupo Brasil AntiSPAM é toda mensagem eletrônica que não segue o código de ética antiSPAM [Grupo Brasil AntiSPAM, 2005], ou seja, que atenda pelo menos dois dos itens a seguir:

- o remetente é inexistente ou possui identidade falsa;
- o destinatário não autorizou previamente o envio da mensagem;
- o destinatário não pode optar em não receber mais a mensagem;
- o assunto não condiz com o conteúdo da mensagem;
- a sigla NS (Não Solicitado) está ausente no assunto de uma mensagem que não foi previamente requisitada;
- o remetente não pode ser identificado;
- uma mensagem semelhante foi recebida anteriormente em menos de dez dias apenas com o remetente ou assunto diferentes.

Nos Estados Unidos, o órgão responsável pela legislação anti-*spam* é o FTC que definiu o estatuto CAN-SPAM (*Controlling the Assault of Non-Solicited Pornography and Marketing Act*) [FTC, 2005] para o controle dos *spams*. O estatuto CAN-SPAM estabelece regras para o envio de e-mails não solicitados. O remetente que não respeitar as regras definidas está sujeito a penas criminais. De acordo com as regras estabelecidas, uma mensagem eletrônica deve conter informações verdadeiras a respeito da sua origem. O assunto da mensagem deve estar relacionado com o próprio conteúdo. Caso seja uma propaganda, uma mensagem eletrônica deve indicar claramente o seu propósito. O endereço físico do remetente deve estar presente na mensagem para possíveis reclamações ou denúncias. Por fim, uma mensagem deve fornecer ao destinatário a opção de não receber mais mensagens semelhantes do mesmo remetente.

[†]O uso da palavra *spam* para denotar mensagens eletrônicas não solicitadas é de origem controversa. A palavra *spam* é uma marca registrada [Hormel Foods, 2000] pela Hormel Foods LLC. A Hormel Foods LLC é uma empresa de alimentos e o nome *spam* pode ter surgido de uma contração das palavras "*SPiced hAM*" que batizou um dos seus produtos. O produto *spam* se tornou conhecido em 1937 durante uma campanha publicitária e em seguida pela utilização durante a segunda guerra mundial pelo exército americano. Como a carne era racionada às tropas, o alimento largamente utilizado era o *spam*. Todos os soldados americanos ao retornar aos EUA recebiam uma medalha que ficou conhecida como medalha *spam*. Como a condecoração foi recebida por diversas pessoas, a palavra *spam* ficou associada a algo comum [Holmes, 2005]. Anos mais tarde, o grupo de comédia Monty Python filmou uma cena onde a palavra *spam* é repetida muitas vezes em pouco tempo. Assim, a palavra *spam* se tornou sinônimo de algo repetitivo e sem sentido como a maioria das mensagens eletrônicas não solicitadas.

A prática de enviar *spam* tem crescido na Internet a ponto de seus usuários manifestarem insatisfação e uma menor credibilidade no correio eletrônico. Os *spams* são de conteúdo variados como informações políticas, religiosas, culturais etc. No entanto, a maioria dos *spams* atuais possui conteúdos publicitários de produtos ou serviços, fraudes e atividades ilícitas. Esta seção descreve como os *spammers* obtêm as listas de endereços, porque é tão fácil enviar *spams*, como os *spammers* mantêm o anonimato e as possíveis medidas para combater o envio de *spams*.

Obtenção de Endereços e a Escolha das Vítimas

A prática do envio de e-mails não solicitados tornou-se uma atividade lucrativa não só com a venda dos produtos e serviços anunciados como também com a própria atividade de criação e envio de *spams*. Muitas empresas pagam *spammers* para criar mensagens eletrônicas para promover as suas ofertas. Outra forma de lucrar com os *spams* é através da obtenção de endereços eletrônicos de usuários da Internet. Para tal, os *spammers* podem utilizar diversas técnicas. Dentre elas estão a invasão de servidores para aquisição de listas de endereços de possíveis membros associados, a interceptação de e-mails de grupos e ainda a utilização de programas populares como o ICQ [ICQ, 2005], o MSN Messenger [Microsoft, 2005] e o Orkut [Büyükkokten, 2005]. Utilizando e-mails de grupos, o *spammer* pode obter diversos endereços a partir de listas em cópia. Conforme o e-mail vai sendo encaminhado, maior é o número de endereços presentes. Já os programas populares citados acima também podem ser usados por *spammers*, pois eles oferecem mecanismos automáticos de busca por nomes e geram listas de endereços associados.

Após obter os potenciais endereços dos destinatários, é necessário saber se eles são ativos. Para saber se um usuário é ativo existem diversas técnicas utilizadas pelos *spammers*. Uma delas consiste em enviar um atalho para um endereço de um sítio (*link*) em uma figura junto com o endereço de e-mail do destinatário conforme a Figura 1.1. Caso o destinatário tente visualizar a figura, ele irá baixá-la do sítio e abrir uma conexão HTTP (*HyperText Transfer Protocol*) com o servidor de imagens do *spammer*. Assim, através dos seus registros de HTTP o *spammer* saberá que o usuário existe, quem ele é e que figura ele visualizou. Esse procedimento permite validar o endereço como ativo e, ao mesmo tempo, criar um perfil associado a este endereço pelo tipo de figura que foi selecionada. Algumas configurações de programas de correio eletrônicos permitem exibir a mensagem automaticamente facilitando a obtenção dos endereços pelos *spammers*. Os *spammers* podem ainda verificar a existência do usuário ao receber a solicitação de não recebimento de um determinado e-mail. Assim, regras estabelecidas por legislações anti-*spam*, como o CAN-SPAM, podem ser utilizadas em benefício do próprio *spammer*. Para evitar esse tipo de rastreamento, um usuário nunca deve visualizar um *spam*.

```
<a href="http://www.meusite.com.br?email=destinatario@email.com.br">
<img src="http://www.meusite.com.br/imagem/figura.gif?email=destinatario@email.com.br"
height=1 width=1 alt=""><\a>
```

Figura 1.1. Técnica para descobrir se o usuário está ativo.

Os valores das listas de endereços aumentam conforme o número de endereços e a

qualidade dos perfis dos destinatários. Assim, *spams* específicos podem ser direcionados para determinadas listas de endereços. Algumas empresas oferecem uma porcentagem sobre cada produto vendido para a pessoa que obteve a lista [Spammer-X et al., 2004]. O perfil do usuário pode ser também obtido através da invasão de servidores específicos, como o servidor de um sítio de astrologia, sítio de jogos, etc.

Evolução do *Spam* e suas Finalidades

A simplicidade do protocolo SMTP (*Simple Mail Transport Protocol*) exige o conhecimento de poucos comandos para se enviar um e-mail, bastando apenas a abertura de uma conexão *Telnet* com um servidor SMTP. Como não é verificado se o usuário que originou o e-mail existe, torna-se fácil gerar um *spam* utilizando outros nomes de remetentes. Essas características facilitaram o surgimento de técnicas para se gerar e enviar e-mails com finalidades lucrativas e ainda manter o anonimato.

Inicialmente, os *spams* eram em maioria e-mails em HTML com uma figura chamativa ou apenas uma frase e um atalho que indicava onde encontrar mais do conteúdo anunciado. Conhecendo as preferências do destinatário, torna-se mais fácil influenciá-lo. A Figura 1.2 mostra um exemplo de um *spam* em HTML.

```
<HTML>
<head><title> Interessado em dinheiro fácil?</title> </head>
<body>
    <a href="http://www.meusite.com.br/dinheiro">
     Clique aqui e entre para o mundo do dinheiro fácil!!! </a>
</body>
</HTML>
```

Figura 1.2. Exemplo de um *spam* em HTML.

Para hospedarem os sítios, os *spammers* utilizavam seus próprios computadores e conexões discadas. Esperava-se que os usuários pagariam a assinatura ao acessar o sítio anunciado pelo *spam*. Desde o princípio, o número de e-mails já era grande o suficiente para garantir que se apenas uma pequena parcela dos destinatários fizesse a assinatura, o lucro já seria grande para o dono do sítio. O número de e-mails não solicitados aumentou rapidamente em virtude das vulnerabilidades da rede. O aumento no número de *spams* gerou a insatisfação dos usuários da rede motivando os desenvolvedores de software a criar programas anti-*spam* e programas adicionais (*plug-ins*) para servidores. As primeiras medidas para evitar os *spams* foram a ameaça do cancelamento das contas dos usuários que enviavam *spams* através de provedores da Internet e a imposição de termos e condições para as empresas promoverem seus produtos. A criação de listas negras (*Realtime Black List* - RBL) também dificulta a hospedagem dos sítios. Se os *spammers* enviarem e-mails com identidade ou IP próprios, eles podem entrar em listas negras. Conseqüentemente, todos os seus e-mails passam a ser considerados *spams* e, portanto, são bloqueados.

Para não entrar em listas negras e não serem encontrados, os *spammers* têm que achar locais onde hospedar os seus sítios e os respectivos arquivos. Se o sítio for hospe-

dado em provedores convencionais, a conta do *spammer* será cancelada dado que denúncias são feitas ao provedor por vítimas de *spams*. Provedores convencionais não permitem que seus assinantes sejam *spammers*, pois podem ser penalizados por infringir políticas anti-*spam*. Assim, os *spammers* utilizam provedores que se propõem a hospedar o conteúdo dos seus sítios. Esses provedores cobram caro por esse serviço e hospedam sítios com qualquer conteúdo ignorando a possibilidade de serem penalizados. Algumas outras técnicas podem ser utilizadas por *spammers* para hospedar seu conteúdo. Os *spammers* podem hospedar seu conteúdo pela Internet em computadores pessoais ou em servidores sem o conhecimento do proprietário. Para tal, é necessário que o computador seja invadido ou que o dono do computador instale inconscientemente programas que dão direitos aos *spammers* de utilizar o computador infectado. Por conseguinte, os *spammers* podem hospedar seu conteúdo gratuitamente.

O surgimento de penas criminais e mecanismos anti-*spams* como as listas negras trouxeram a necessidade de se manter o anonimato no envio dos *spams*. O anonimato é importante visto que os *spams* podem conter informações falsas, oferecer produtos inexistentes e ainda utilizar identidades de terceiros configurando em atividades sujeitas a penas criminais. Para evitar também que o seu endereço IP ou domínio seja bloqueado por outros servidores de e-mail, é necessário que o *spammer* mantenha o anonimato e utilize outros computadores para enviar os seus *spams*.

A primeira forma de enviar *spams* mantendo o anonimato foi através de servidores *proxy* transparentes que utilizam protocolos como o Socks v5 [Leech, 1996]. Utilizando o Socks v5 em sua configuração padrão, um *proxy* transparente permite que qualquer computador na Internet o utilize para encaminhar seus pacotes para qualquer servidor de e-mails, sem se identificar. Para evitar que esse *proxy* seja rapidamente adicionado a listas negras, o *spammer* utiliza alternadamente diversos servidores *proxy* vulneráveis para enviar os seus e-mails. Outra estratégia empregada para enviar *spams* é utilizar servidores em países com línguas diferentes para dificultar a comunicação entre o servidor de e-mail atacado e o responsável pelo *proxy* utilizado.

Utilizar servidores SMTP com *relay* aberto é uma outra forma de enviar *spams*. Ao enviar um e-mail, o *spammer* utiliza o servidor SMTP com *relay* aberto para reencaminhar o seu e-mail mantendo o anonimato, de forma similar ao envio por servidores *proxy*. O emprego de *relays* SMTP era bastante comum nos provedores de Internet permitindo que qualquer usuário da Internet os acessassem.

Para não serem usados por *spammers*, tanto os servidores *proxy* quanto os SMTP não devem encaminhar fluxos de dados de computadores externos à sua rede local e devem utilizar *firewalls*. É importante notar que os endereços IP de origem não podem ser forjados porque o SMTP funciona sobre o TCP (*Transmission Control Protocol*), portanto é necessário o estabelecimento de conexão.

Atualmente, servidores *proxy* e servidores SMTP com *relay* aberto não são tão utilizados por *spammers*. Assim, como os *spammers* conseguem facilmente encontrá-los na rede, eles também são rapidamente adicionados em listas negras de provedores da Internet. Algumas técnicas que também podem manter o *spammer* anônimo são exploradas. Os *spams* podem ser enviados através de programas de bate-papo como o ICQ ou o MSN Messenger por usuários maliciosos. Computadores que utilizam faixas de endereços IP

dinâmicos também podem ser utilizados para envio de e-mails não solicitados, pois garantem o anonimato da fonte. Diferentes endereços IP dentro de uma faixa são fornecidos aos usuários de grandes provedores na Internet cada vez que ele se conecta. Assim, não há como garantir que um determinado usuário é realmente o *spammer*. Para combater essa prática, é necessário bloquear faixas de endereços IP ou domínios. Porém, nem sempre essa atitude é possível porque na maioria das vezes essa faixa pertence a algum grande provedor de acesso.

Uma técnica elaborada de envio de *spams* consiste em se apropriar de um roteador e de uma faixa de endereços IP. Os *spammers* adquirem faixas de endereços IP válidos que não estejam sendo utilizados e invadem um roteador com falhas de segurança. Assim, ele manipula o pedido de atualização de rotas e anuncia a todos os outros roteadores vizinhos que o roteador invadido é a única rota para a faixa de endereços IP adquirida maliciosamente. O *spammer* configura o roteador para transferir todo os fluxos de dados gerados a partir de *spams* para o seu computador, toma posse de uma faixa de endereços IP e de um roteador para uso próprio. Quando um dos endereços IP da faixa é adicionado a uma lista negra, ele passa a utilizar outro endereço IP da faixa adquirida. Como não há relação entre endereços IP e regiões geográficas, é dificil de se localizar o roteador invadido a partir do seu endereço IP.

Mecanismos Anti-*spam*

Para se evitar *spams*, muitas empresas investem no desenvolvimento de mecanismos anti-*spam*. Existem diversas formas de se evitar os *spams*, porém todas elas se baseiam na filtragem de mensagens não solicitadas. A filtragem pode ser realizada de acordo com os campos do cabeçalho e o conteúdo de um e-mail.

A filtragem pelo o cabeçalho explora principalmente os campos de endereço de origem, nome do remetente e assunto do e-mail. Um usuário pode configurar seu cliente de e-mail para filtrar todas as mensagens provenientes de endereços ou remetentes indesejados. Além dessa filtragem, os programas anti-*spam* podem ainda verificar a consistência das informações contidas no cabeçalho. Por exemplo, se o endereço IP de origem não condiz com o domínio, esse e-mail deve ser filtrado. Se o e-mail tiver sido encaminhado por um *proxy*, o e-mail também pode ser bloqueado devido às conhecidas vulnerabilidades.

A consistência de informações contidas no cabeçalho pode ser verificada através do DNS (*Domain Name System*) reverso. O DNS reverso deve estar sempre configurado para apontar o endereço IP correspondente. Como os *spammers* devem manter o anonimato, o DNS reverso de seus computadores não são configurados, ou seja, eles não apontam o seu endereço IP. Os programas anti-*spam* bloqueiam então todo o e-mail cujo remetente não possui DNS reverso. Muitos provedores de acesso utilizam faixas de endereços IP dinâmicos e, por conseguinte, seus clientes utilizam um DNS reverso genérico adotado pelo provedor. Como utilizar endereços IP dinâmicos é uma forma de manter anonimato usado por *spammers*, os anti-*spams* normalmente bloqueiam esse tipo de conexão. Um dos problemas dessa abordagem é que diversos administradores de redes não configuram o DNS reverso, gerando falsos positivos.

Os programas anti-*spam* podem também utilizar listas para fazer filtragem por ca-

beçalho. Existem três tipos de listas: as negras, as cinzas e as brancas. As listas negras são utilizadas para bloqueio incondicional dos e-mails. Sempre que um e-mail é recebido proveniente de um endereço ou domínio, contido numa lista negra, ele é bloqueado. Os domínios ou endereços IP são adicionados às listas negras após se verificar um alto número de *spams* provenientes dessa fonte. Já os programas que utilizam listas cinzas (*Gray Lists*) bloqueiam sempre toda a primeira conexão SMTP desconhecida e enviam ao servidor SMTP do remetente uma mensagem de erro temporária. Caso o servidor de e-mail seja legítimo, ele irá reenviar mais tarde o e-mail. Para evitar a recepção de milhares de mensagens de erro, alguns *spammers* não utilizam servidores SMTP. São usados clientes que se conectam diretamente ao servidor SMTP do destinatário. Como os *spammers* não recebem as mensagens de erro temporárias, eles não retransmitem e-mails bloqueados por listas cinzas. As listas brancas (*White Lists*) são listas que contêm os nomes dos servidores e domínios confiáveis. Sempre que um e-mail é recebido proveniente de algum dos endereços IP ou domínios pertencentes à lista branca, ele é aceito.

Caso o assunto do e-mail possua palavras exploradas em e-mails comerciais, esses serão também filtrados.

O bloqueio por conteúdo é realizado através da detecção de palavras comerciais comuns no corpo dos e-mails, reconhecimento de padrões ou pela presença de anexos suspeitos. E-mails com palavras como “Viagra” são facilmente bloqueadas, pois na maioria das vezes não são requisitados. Uma forma de burlar um sistema anti-*spam* é mudar letras de palavras conhecidas como “Viagra” por outras com caracteres semelhantes como “\/iagra”. Uma outra forma de burlar o bloqueio por conteúdo é colocar o anúncio numa figura, assim o corpo e o assunto do e-mail não contêm nenhuma evidência procurada. Alguns anti-*spams* são configurados para sempre bloquear arquivos anexados porque o anúncio pode estar disfarçado como figura ou algum outro tipo de arquivo.

Outra forma de filtragem por conteúdo é através da análise de diversos e-mails enviados para destinatários distintos. Caso um padrão semelhante seja observado, há uma grande probabilidade de ser um e-mail não solicitado enviado a diversas pessoas. Como a maioria dos filtros por conteúdo é determinística, adicionando-se aleatoriedade a um *spam* evita-se que um comportamento ou conteúdo comum seja detectado. Para tornar os anti-*spams* adaptativos, mecanismos baseados em técnicas de inteligência artificial são empregados para filtrar os *spams* de acordo com o seu conteúdo.

Aproximações baseadas em técnicas de inteligência artificial usando redes Bayesianas para classificar e-mails vêm sendo propostas [Sahami et al., 1998]. Através das redes Bayesianas é possível a adaptação da classificação do e-mail ao longo da utilização do anti-*spam*. Baseado numa lista de regras e em experiências passadas é atribuída uma probabilidade de um e-mail ser um *spam* a cada uma das regras. Cada e-mail é comparado com cada regra para decidir se o e-mail se trata de um *spam*. Ao final da análise, a probabilidade do e-mail ser um *spam* é computada baseada em todas as comparações. As probabilidades de cada regra são atualizadas a cada e-mail de acordo com a ação do usuário, isto é, se um usuário exclui ou não o e-mail.

Os anti-*spams* podem tornar-se mais eficientes se aplicarem diversas técnicas em conjunto. Esse é o princípio de programas como o Spam-Assassin [SpamAssassin, 2005] que utiliza técnicas de filtragem por cabeçalho e por conteúdo. Novos conceitos baseados

em confiança podem ser aplicados aos programas anti-*spam*. O conceito de confiança reflete o quanto se acredita no remetente ou no endereço IP de origem do e-mail. A confiança pode então se tornar uma métrica dinâmica atualizada ao longo da operação do mecanismo anti-*spam*. Conforme se recebe e-mails de um mesmo remetente ou domínio, avalia-se o número de *spams* recebidos. Caso esse número seja alto, a confiança é baixa; do contrário, a confiança é alta. Cabe a cada usuário definir qual é o grau de confiança necessário para receber os e-mails.

1.3. Negação de Serviço

Diferentemente da maioria dos ataques da Internet, um ataque de negação de serviço não visa invadir um computador para extrair informações confidenciais, como números de cartões de crédito e senhas bancárias, e nem para modificar o conteúdo armazenado neste computador, como sítios da Internet. Tais ataques têm como objetivo tornar inacessíveis os serviços providos pela vítima a usuários legítimos. Nenhum dado é roubado, nada é alterado e não ocorre nenhum acesso não autorizado ao computador da vítima. A vítima simplesmente pára de oferecer o seu serviço aos clientes legítimos, enquanto tenta lidar com o tráfego gerado pelo ataque [Mirkovic et al., 2004]. Um serviço pode ser o uso de um buscador de páginas, a compra de um determinado produto ou simplesmente a troca de mensagens entre duas entidades. O resultado de um ataque de negação de serviço pode ser o congelamento ou a reinicialização do programa da vítima que presta o serviço, ou ainda o esgotamento completo de recursos necessários para prover o seu serviço.

Os primeiros problemas com ataques de negação de serviço na Internet surgiram no início dos anos 90. Nesse período, foram desenvolvidos os primeiros programas para ataques remotos. Para usar estes programas e obter um grande impacto, o atacante precisava de um computador de grande capacidade conectado a uma rede de alta velocidade. Esses dois requisitos são encontrados em computadores de universidades, o que acarretou em um grande número de roubo de senhas de usuários nestas instituições. Hoje, os ataques de negação de serviço são comuns e os prejuízos financeiros e de imagem que eles causam atingem cifras enormes. Sem sombra de dúvida, o estudo e o desenvolvimento de técnicas contra ataques de negação de serviço tornaram-se importantes temas na área de segurança.

Nesta seção são apresentadas as formas existentes de negação de serviço. São abordadas ainda as características da arquitetura da Internet que possibilitam a execução destes ataques, bem como são apresentados os principais fatores que motivam os atacantes a interferir em um serviço. Por fim, alguns dos principais ataques de negação de serviço são caracterizados e classificados.

1.3.1. Formas de Negação de Serviço

Uma forma comum de prejudicar a provisão de um determinado serviço é através do consumo de recursos essenciais para o seu funcionamento, como a memória, o processamento, o espaço em disco e a banda passante. Como estes recursos são limitados, sempre é possível inundar a vítima com um grande número de mensagens de forma a consumir quase que a totalidade dos seus recursos. Este tipo de ataque é denominado “ataque por inundação”. Nestes ataques, a aplicação, a vítima ou a própria infra-estrutura de rede

fica sobrecarregada com o tratamento das mensagens de ataque e não consegue atender ao tráfego de usuários legítimos.

Um dos fatores que dificulta a adoção de medidas contrárias aos ataques de inundação é o fato de não ser possível distinguir o tráfego de ataque do tráfego de usuários legítimos. Ou seja, os atacantes utilizam mensagens normalmente empregadas em comunicações convencionais. Desta forma, ações que priorizem o tráfego legítimo em detrimento do tráfego de ataque são difíceis de serem tomadas.

Para que um ataque de inundação seja bem sucedido, um atacante deve gerar mensagens a uma taxa superior à taxa na qual a vítima, ou a sua infra-estrutura de rede, consegue tratar estas mensagens. Este é um obstáculo para os ataques de inundação. Em vítimas com poucos recursos, é mais fácil obter o sucesso da negação de serviço, uma vez que é necessário um esforço menor para tornar o serviço indisponível. No entanto, sendo a vítima superdimensionada, como é o caso de servidores mais famosos, o sucesso de um ataque não é alcançado facilmente. Muitas vezes, o tráfego gerado por apenas uma estação de ataque não é suficiente para exaurir os recursos da vítima. Nestes casos, são necessárias diversas estações atacando em conjunto para que os recursos da vítima se esgotem. Quando diversas estações agem com um objetivo comum de atacar uma determinada vítima, tem-se um ataque de negação de serviço distribuído (*Distributed* DoS - DDoS).

Os ataques distribuídos são uma ameaça mesmo para vítimas superdimensionadas. Sempre é possível inundar uma vítima e negar o seu serviço desde que um número suficiente de estações participe do ataque. Alguns sítios famosos como Yahoo, Ebay, Amazon.com e CNN.com já foram alvos de ataques de negação de serviço distribuídos bem sucedidos [Garber, 2000], [CNN.com, 2000]. Hoje em dia, diversas ferramentas que automatizam ataques distribuídos podem ser encontradas na Internet [Dittrich, 1999a], [Dittrich, 1999c], [Dittrich, 1999b]. Com isso, o nível de conhecimento necessário para se iniciar ataques distribuídos é reduzido drasticamente, fazendo com que atacantes inexperientes consigam inundar vítimas com poucos comandos.

Uma segunda maneira de se negar um serviço é através da exploração de alguma vulnerabilidade existente na vítima, os chamados “ataques por vulnerabilidade”. Ao invés de inundar a vítima de forma a consumir seus recursos, tais ataques se aproveitam de determinadas vulnerabilidades para tornar seus serviços inacessíveis. Na maioria dos casos, estas vulnerabilidades são o resultado de falhas no projeto de determinada aplicação, do próprio sistema operacional ou até mesmo de um protocolo de comunicação [Watson, 2004], [Gont, 2004]. Diversas vulnerabilidades deste gênero têm sido identificadas e exploradas. Um caso particular dos ataques por vulnerabilidades são aqueles que exigem somente um pacote para causar a negação do serviço [CERT, 1996], [Gont, 2004], [Cisco, 2003], [US-CERT, 2005]. Neste caso, ao receber um pacote contendo determinadas características que ativam a vulnerabilidade, a vítima se encontra em uma situação imprevista e reage diferentemente em cada caso. Como resultado, o serviço pode ser indefinidamente interrompido até que uma intervenção manual ocorra. Através do envio periódico do mesmo pacote para a vítima, tais ataques podem ser prolongados até que um método de correção da vulnerabilidade seja divulgado. Estes ataques apresentam um grande desafio para sistemas de defesa, especialmente para os sistemas de

rastreamento, uma vez que estes dispõem de apenas um pacote para identificar o atacante.

1.3.2. A Negação de Serviço e a Arquitetura da Internet

Três características são responsáveis pelo sucesso da Internet: a velocidade de transmissão de dados, a confiabilidade e o compartilhamento do meio com a conseqüente redução do custo de comunicação. Tais características provêm da técnica de comutação por pacotes e dos protocolos TCP/IP (*Transmission Control Protocol*/*Internetworking Protocol*). Entretanto, alguns princípios da comutação de pacotes e dos protocolos TCP/IP facilitam a ploriferação de ataques de negação de serviço.

Na Internet não há reserva de recursos em uma comunicação entre dois nós, pois se adota o "melhor esforço" como procedimento básico. Tal fato faz com que um ataque de negação de serviço busque consumir o máximo dos recursos da rede. Dessa forma, os usuários legítimos são prejudicados, uma vez que os recursos, ora usados por eles, estão sendo empregados em ações maliciosas.

O roteamento é outro fator facilitador para os ataques de negação de serviço. Na Internet, um pacote pode ser encaminhado através de qualquer rota entre o nó fonte e o nó destino. O processo de seleção de rotas é feito pelos nós intermediários e é transparente para a fonte e para o destino. Além da transparência na seleção de rotas, somente o endereço de destino é usado pela infra-estrutura de rede para que um pacote seja devidamente roteado até o seu destino. Nenhuma verificação é realizada para confirmar a autenticidade dos pacotes IP, ou seja, não existem métodos para assegurar que o endereço de origem de cada pacote IP é autêntico. Desta forma, atacantes experientes que não necessitam de comunicações bidirecionais podem se aproveitar desta característica para manter o anonimato durante ataques de negação de serviço. Tanto em ataques por inundação quanto em ataques por vulnerabilidades, atacantes podem forjar o endereço IP de origem e enviar tais pacotes para a vítima sem deixar nenhum rastro que possa identificá-los.

Como a seleção de rotas é feita pelos nós intermediários, torna-se difícil detectar e filtrar pacotes IP com endereço de origem forjado. A Figura 1.3 ilustra esse problema. Os pacotes gerados pelo nó *A* e destinados ao nó *B* são encaminhados através do roteador R_1. O roteador R_2 está localizado em outro ponto da Internet. Quando um pacote com endereço de origem do nó *A* e destinado ao nó *B* é recebido por R_2, este nó não consegue determinar se o endereço da fonte contido no pacote é forjado, ou se o roteador R_1 falhou e, por isso, os pacotes estão sendo encaminhados por outra rota. Dessa forma, o roteador R_2 encaminha normalmente o pacote para o nó *B*.

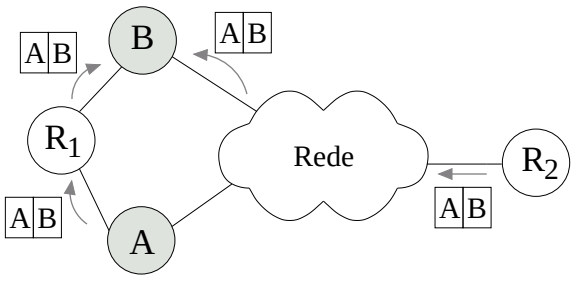

Figura 1.3. O problema do roteamento e da autenticidade dos endereços IP de origem.

A topologia da Internet também contribui para os ataques de negação de serviço. O núcleo da rede é composto por enlaces de alta capacidade. Por outro lado, os nós nas bordas são normalmente providos de enlaces de baixa capacidade e que estão conectados ao núcleo. Esta assimetria permite que os nós localizados no núcleo encaminhem pacotes de diversas origens distintas para diferentes destinos. Como resultado, ela também possibilita que os nós localizados nas bordas sejam inundados pelo tráfego agregado de outros nós. Isto é exatamente o que ocorre durante um ataque de negação de serviço distribuído. Para atacar uma determinada vítima, cada nó gera somente uma pequena parcela do tráfego agregado. Este tráfego gerado pode nem ser detectado como uma anomalia em determinadas redes e passar desapercebido pelos roteadores.

A arquitetura da Internet também dificulta a adoção de mecanismos de defesa contra ataques de negação de serviço. Na Internet, para garantir requisitos como a escalabilidade e o custo reduzido, a complexidade está localizada nos nós de borda da rede. Tal fato provém do modelo de melhor esforço, adotado para o encaminhamento simples dos pacotes na camada de rede, e do paradigma fim-a-fim, segundo o qual requisitos específicos da aplicação devem ser garantidos pelos protocolos da camada de transporte e de aplicação nos usuários finais. Por estes motivos, mecanismos de defesa localizados na infra-estrutura de rede são criticados, como será visto na Seção 1.4.

A forma como a Internet é gerenciada também cria desafios adicionais no combate aos ataques de negação de serviço. Não há hierarquia. A Internet é formada por uma comunidade de redes, interconectadas para prover acesso global aos seus usuários [Mirkovic et al., 2004]. Cada uma das redes que compõe a Internet é gerenciada localmente, de acordo com políticas definidas por seus proprietários. Neste sentido, a adoção de mecanismos de defesa contra ataques de negação de serviço, que necessitam operar em um grande número de nós da rede, é prejudicada. Como não existe uma entidade central, não há como garantir a adoção de tal sistema em larga escala.

1.3.3. A Motivação para os Atacantes

Diversos fatores, positivos ou negativos, podem motivar um ataque de negação de serviço. Um usuário bem-intencionado pode iniciar um ataque para provar que é possível inundar a vítima ou explorar alguma de suas vulnerabilidades. Em caso de sucesso, pode-se tentar encontrar uma solução para evitar que ataques originados de usuários mal-intencionados aconteçam. Uma vez encontrada a solução, ela pode ser amplamente divulgada para acelerar o processo de proteção.

Os atacantes também podem agir em busca de reconhecimento na comunidade virtual. Um atacante que consegue forçar um sítio bem conhecido a sair do ar ganha uma certa visibilidade e é bem considerado pelos colegas, podendo inclusive ser admitido em um determinado grupo de ataque de mais alto nível. Por outro lado, com a automação de fases importantes de ataques de negação de serviço distribuídos e o fácil acesso a estes procedimentos [Mirkovic e Reiher, 2004], tal tarefa ficou ainda mais fácil. Agora, até mesmo atacantes com pouca experiência são capazes de deixar suas vítimas inacessíveis, ganhando a reputação desejada.

Um ataque de negação de serviço também pode ser motivado por lucro financeiro. Um usuário com habilidade suficiente para iniciar um ataque pode deixar um determinado

servidor “fora do ar” por um certo período em troca de uma remuneração paga por terceiros. Uma empresa poderia contratar tais “serviços” e usar estes ataques para prejudicar a imagem de seus concorrentes e se beneficiar diretamente. Este procedimento pernicioso alcança requintes de organização inimagináveis, pois existem redes inteiramente formadas por computadores comprometidos, os chamados zumbis, que estão sendo alugadas de forma a serem usadas em ataques de negação de serviço [Reuters, 2004]. O trabalho de comprometer um determinado número de estações e controlá-las remotamente é realizado inicialmente e depois os recursos disponíveis em cada uma destas estações é colocado à disposição de quem quiser alugá-los para qualquer tipo de atividade, como o envio de *spams*, fraudes digitais e ataques de negação de serviço. Outro exemplo assustador é o uso destes ataques como forma de extorsão. Uma maneira que vem sendo usada freqüentemente por atacantes é exigir um pagamento de determinados sítios para evitar que seu servidor seja atacado [Shachtman, 2003]. Representantes de organizações que fazem o pagamento têm o sítio “assegurado” enquanto aqueles que não pagam sofrem consecutivos ataques de curta duração.

Motivos políticos também podem ser a causa de um ataque de negação de serviço. Um determinado grupo com idéias diferentes de uma determinada organização ou instituição pode atacar a sua rede e os seus servidores de modo a impedir a divulgação de certas informações. Como exemplo, o sítio da rede de televisão Al-Jazeera sofreu um intenso ataque de negação de serviço distribuído durante a investida americana no Iraque [Gray e Fried, 2003]. Aparentemente, o ataque foi direcionado aos servidores DNS, cuja função é converter o nome do sítio no seu respectivo endereço IP, usados pela emissora. Como conseqüência, não era possível determinar o seu endereço IP e o acesso era negado.

1.3.4. Classificação dos Ataques

Diversos fatores, como o número de atacantes envolvidos e o tipo de recurso explorado na vítima, podem ser usados para classificar os ataques de negação de serviço. Nesta seção, os ataques são classificados em: por inundação, por refletor, a infra-estrutura de redes, por vulnerabilidades e distribuídos.

Ataques por Inundação

Um dos ataques de negação de serviço mais conhecidos é o ataque por inundação de segmentos TCP SYN, que explora o procedimento de abertura de conexão do protocolo de transporte TCP. O protocolo TCP é utilizado em serviços que necessitam de entrega confiável de dados, como a transferência de arquivos. Uma conexão TCP se inicia com a negociação de determinados parâmetros entre o cliente e o servidor. A Figura 1.4 ilustra o processo de abertura de uma conexão TCP. Inicialmente, o cliente envia um segmento TCP SYN para o servidor, indicando um pedido de abertura de conexão. Este segmento leva consigo um parâmetro denominado número de seqüência inicial. O número de seqüência possibilita que o receptor reconheça dados perdidos, repetidos ou fora de ordem. Após o recebimento do segmento TCP SYN, o servidor necessita de tempo para processar o pedido de conexão e alocar memória para armazenar informações sobre o cliente. Em seguida, um segmento TCP SYN/ACK é enviado como resposta de forma a notificar o cliente que o seu pedido de conexão foi aceito. O segmento de resposta reco-

nhece o número de seqüência do cliente e envia o número de seqüência inicial do servidor. Por fim, o cliente envia um segmento TCP ACK para reconhecer o número de seqüência do servidor e completar a abertura da conexão. Esse procedimento é conhecido como o acordo de três vias (*three-way handshake*).

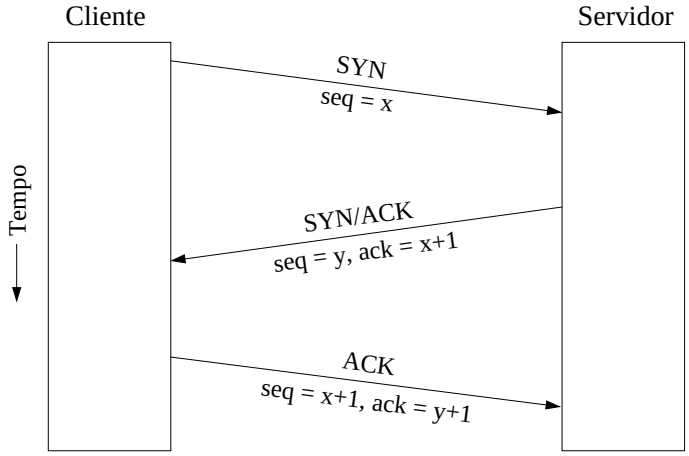

Figura 1.4. Abertura de uma conexão TCP e o acordo de três vias.

A partir deste procedimento, os recursos da vítima podem ser explorados de maneiras distintas. Para analisar com mais detalhes como cada recurso é explorado, são definidos alguns parâmetros relacionados à vítima e ao atacante. Seja t_a o intervalo de tempo entre cada segmento TCP SYN enviado pelo atacante, t_p o tempo necessário para que a vítima processe um pedido de conexão TCP e envie uma resposta, e t_m o tempo em que um determinado recurso de memória fica alocado para uma conexão TCP.

Um ataque ao processamento da vítima pode ser realizado quando o atacante consegue gerar segmentos a uma taxa mais rápida do que a vítima consegue processá-los, ou seja, quando $t_a < t_p$. A Figura 1.5 mostra um ataque de negação de serviço realizado para sobrecarregar o processamento da vítima. Neste caso, a vítima não consegue processar os pedidos de conexão em tempo hábil, o que faz com que a fila de pedidos encha e que muitos deles sejam descartados. Desta forma, caso um usuário legítimo tente acessar o serviço que está sendo atacado, é muito provável que seu pedido de conexão seja descartado junto com o tráfego de ataque. Isso ocorre porque o tráfego do usuário legítimo precisa disputar o mesmo recurso com os inúmeros segmentos enviados pelo atacante. É importante ressaltar ainda que, para permanecer anônimo, o atacante não precisa usar seu endereço IP verdadeiro para realizar o ataque. Na verdade, qualquer endereço IP pode ser usado como endereço de origem nos pacotes de ataque. O uso de endereços de origem forjados não altera em nada o efeito sofrido pela vítima. No entanto, a resposta ao pedido de conexão não retorna para o atacante, mas sim para o endereço usado em seus pacotes. Desta forma, o atacante consegue não só negar o serviço da vítima, mas também permanecer anônimo.

Um outro recurso que pode ser explorado durante ataques por inundação de segmentos TCP SYN é a memória da vítima. Quando o servidor recebe um pedido de abertura de conexão TCP, ele aloca uma pequena quantidade de memória para armazenar determinadas informações sobre o estado da conexão, como os números de seqüência ini-

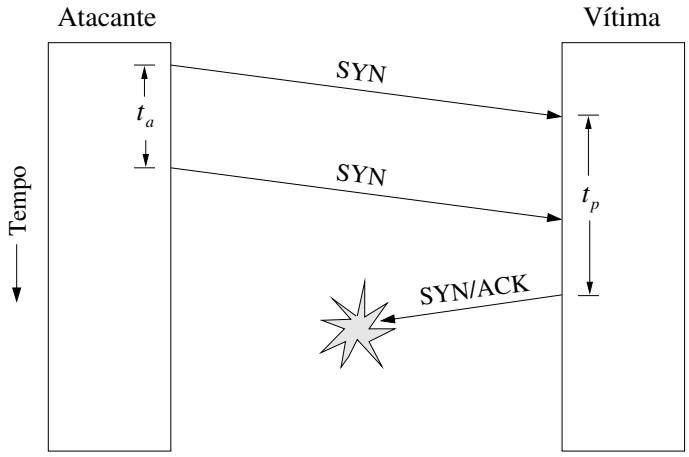

Figura 1.5. Ataque de negação de serviço consumindo o processamento da vítima.

ciais, os endereços IP e as portas TCP usadas nesta conexão. Em seguida, um segmento TCP SYN/ACK é enviado de volta para o cliente de forma a finalizar o acordo em três vias. Neste ponto, é dito que a conexão TCP se encontra semi-aberta. Os recursos reservados inicialmente ficam então alocados até que a conexão seja finalizada por meios convencionais, ou até que um temporizador estoure indicando que o esperado segmento TCP ACK não foi recebido. Neste caso, a memória alocada é finalmente liberada para ser aproveitada em conexões futuras. Em um ataque visando esgotar a memória, o atacante forja o endereço IP de origem dos pacotes, substituindo-o por algum endereço não utilizado. Estes pacotes são então enviados para a vítima de forma a iniciar simultaneamente diversas conexões semi-abertas, consumindo a sua memória. Para que este tipo de ataque tenha sucesso, é necessário que o atacante consiga gerar segmentos a uma taxa maior do que a vitima consegue liberar recursos para as novas conexões, ou seja, é preciso que $t_a < t_m$. A Figura 1.6 ilustra o caso em que o recurso atacado é a memória da vítima. Nesta situação, a mesma disputa entre o tráfego legítimo e o tráfego de ataque ocorre na vítima. À medida que uma conexão libera sua memória, um outro pedido de abertura de conexão é processado. Devido ao volume de tráfego de ataque enviado para a vítima, muito provavelmente este novo pedido processado não é legítimo e só serve para ocupar a memória da vítima por um determinado período. Quando a memória se esgota, novos pedidos de abertura de conexão não são atendidos. É importante observar que, mesmo com limites de memória controlados pelo sistema operacional, a negação de serviço ocorre quando a memória destinada para as conexões se esgota, sem ser necessário o esgotamento completo de toda a memória da vítima.

Nota-se, por definição, que a desigualdade $t_m > t_p$ é sempre válida. Desta forma, três possibilidades podem ocorrer. A primeira possibilidade ocorre quando $t_a < t_p < t_m$, ou seja, o intervalo entre o envio de pacotes de ataque é menor que o tempo de processamento da vítima. Este é o caso abordado anteriormente onde o processamento da vítima é sobrecarregado. Vale também ressaltar que, além do processamento, a memória da vítima também é sobrecarregada. Isto ocorre porque os pedidos de conexão que a vítima consegue atender a tempo, alocam um espaço de memória que só é liberado quando o temporizador estourar, uma vez que pacotes com endereços de origem forjados são usados. A segunda

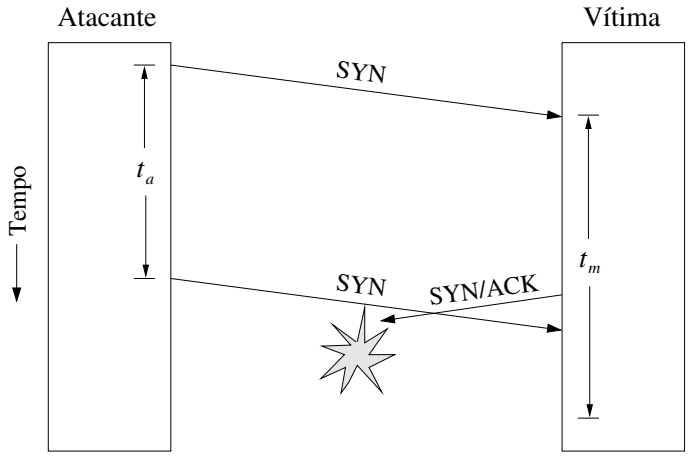

Figura 1.6. Ataque de negação de serviço consumindo a memória da vítima.

possibilidade ocorre quando $t_p \leq t_a < t_m$. Neste caso, a vítima consegue processar as requisições, mas espaços de memória vão sendo consumidos por cada nova conexão e não são liberados a tempo para atender a todas as conexões. Portanto, novas conexões não são aceitas por falta de memória. O último caso ocorre quando o atacante não consegue gerar pacotes de ataque a uma taxa suficiente nem para sobrecarregar a memória, ou seja, $t_a \geq t_m$. Neste caso, os recursos da vítima não são atacados diretamente e outras formas de ataques são necessárias para negar o serviço oferecido.

Uma solução proposta por Schuba *et al.* [Schuba et al., 1997] sugere que o servidor não armazene nenhum estado ao receber um segmento TCP SYN, de forma a evitar ataques por inundação que sobrecarreguem a memória. A idéia dos autores é que somente um segmento TCP SYN/ACK seja enviado como resposta e que o número de seqüência enviado pelo servidor seja o valor *hash* dos endereços IP de origem e destino, das portas TCP, do número de seqüência inicial do cliente e de um valor secreto armazenado no servidor. Ao receber o segmento TCP ACK, o servidor verifica se o seu número de seqüência está correto através do cálculo do valor *hash*. Caso esteja correto, a memória é enfim alocada para a conexão. Apesar de ser eficiente contra ataques que visam esgotar a memória, este mecanismo apresenta pequenas desvantagens. Um problema é a possibilidade de conexões serem estabelecidas somente com o segmento TCP ACK. Apesar de a probabilidade deste caso ocorrer seja pequena, tal evento não é impossível. Além disso, o protocolo TCP não consegue mais oferecer a tolerância a falhas para o caso de conexões semi-abertas, uma vez que não há mais registro destas conexões no servidor.

É importante ressaltar que o consumo de processamento e memória da vítima através de ataques de inundação não ocorre somente em ataques ao protocolo TCP. Na verdade, o conceito apresentado aqui é mais geral e pode ser aplicado a outros protocolos usados. O ataque ao processamento da vítima pode ser realizado praticamente a qualquer protocolo, uma vez que todas as mensagens recebidas precisam ser processadas. O ataque à memória, por outro lado, exige algum tipo de alocação de recursos de memória na vítima para que tenha sucesso.

Ataques por Refletor

Um outro tipo de ataque de negação de serviço conhecido é o ataque por refletor. Este ataque também é um ataque por inundação que visa consumir recursos da vítima. Porém, devido a presença de uma estação intermediária entre o atacante e a vítima, ele é aqui tratado como um ataque diferenciado. A idéia é usar a estação intermediária para refletir o tráfego de ataque em direção à vítima. Tal manobra dificulta ainda mais a descoberta da identidade dos atacantes, uma vez que o tráfego que chega à vítima é originado no refletor, e não no próprio atacante. Para a reflexão do tráfego de ataque, é necessário que o atacante envie algum tipo de requisição (REQ) para o refletor, usando como endereço de origem o endereço da vítima ao invés do seu próprio endereço. Ao receber uma requisição, o refletor não consegue verificar a autenticidade da origem dessa requisição e, conseqüentemente, envia uma resposta (RESP) diretamente para a vítima. A Figura 1.7 mostra este procedimento de maneira sucinta.

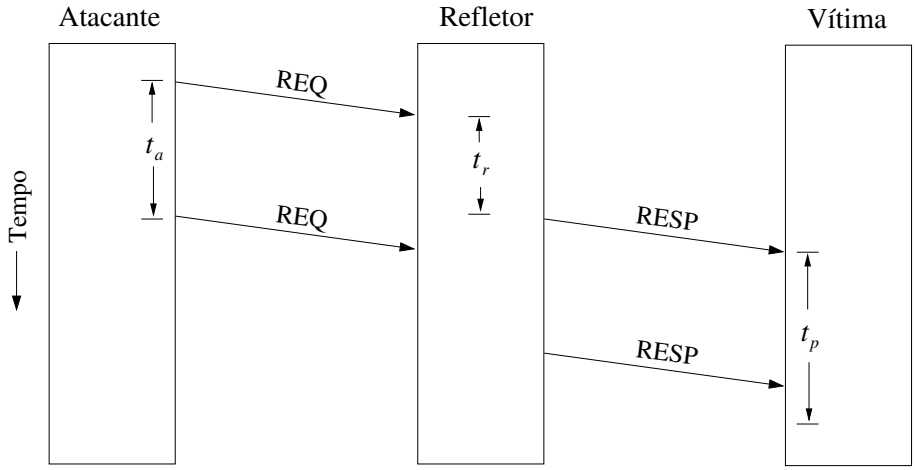

Figura 1.7. Ataque de negação de serviço por um refletor.

Para que o processamento da vítima seja sobrecarregado, é necessário que $t_a < t_p$. Entretanto, como o objetivo do ataque é usar os recursos do refletor e não inundá-lo, é preciso que o refletor consiga processar as requisições a tempo. Desta forma, o tempo de processamento do refletor t_r precisa ser menor ou igual ao intervalo entre pacotes de ataque, ou seja, $t_r \leq t_a$. Caso contrário, o processamento do refletor será sobrecarregado e o tráfego excedente será descartado, não apresentando efeito na vítima. Portanto, o ideal para o atacante é gerar os pacotes de ataque de acordo com a desigualdade $t_r \leq t_a < t_p$.

É importante ressaltar que este tipo de ataque não é restrito a um determinado protocolo. Para o seu funcionamento, é necessário somente um protocolo qualquer que envie um pacote de resposta ao atender a algum tipo de requisição. Desta forma, o próprio protocolo TCP abordado na seção anterior pode ser usado para tal finalidade. No caso, o atacante envia diversos segmentos TCP SYN para o refletor, que os responde com segmentos TCP SYN/ACK direcionados para a vítima. Outra possibilidade é o uso do protocolo UDP (*User Datagram Protocol*) associado ao serviço de DNS. Neste caso, o atacante pode enviar diversas requisições ao serviço de DNS do refletor, que envia uma resposta para a vítima.

Uma das vantagens deste tipo de ataque é que o próprio refletor pode contribuir para o consumo de recursos da vítima. Isso ocorre quando uma mensagem de requisição enviada pelo atacante é menor que a mensagem de resposta enviada pelo refletor. Neste caso, é dito que o refletor também atua como amplificador do tráfego de ataque. Um exemplo típico onde o tráfego enviado pelo atacante é amplificado é o ataque Smurf [CERT, 1998], ilustrado na Figura 1.8. Neste caso, o atacante envia pacotes ICMP para o endereço IP de difusão de uma determinada rede usando o endereço IP de origem da vítima. Como conseqüência, todos as estações daquela rede respondem à requisição, enviando uma resposta para a vítima. Neste caso, todas as estações da rede foram usadas como refletores apenas com o envio de um único pacote.

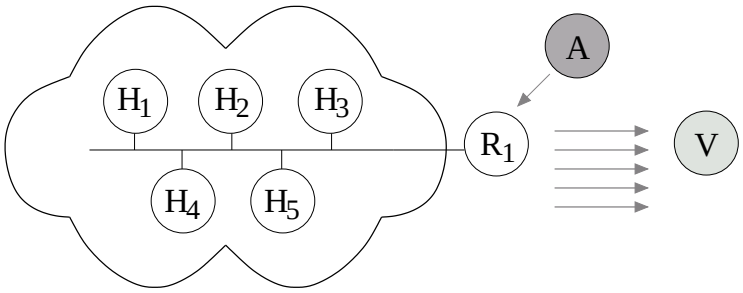

Figura 1.8. Ataque através da inundação ICMP: o atacante A envia pacotes `ping` para a rede refletora se passando pela vítima V para provocar uma inundação endereçada à vítima.

Outro exemplo de ataque por refletor inunda as vítimas com pacotes UDP. Neste caso, um atacante pode deixar fora do ar duas vítimas, fazendo com que as duas enviem continuamente pacotes uma para a outra. Para que tal fato ocorra, o atacante deve enviar um pacote aparentemente originado pelo serviço ICMP `echo` da vítima V_1 para o serviço de geração de caracteres (*chargen*) [Postel, 1983] da vítima V_2. Como mostra a Figura 1.9, ao receber o pacote ICMP `echo` com endereço de origem de V_1, a vítima V_2 envia uma resposta para a vítima V_1, que responde novamente para V_2 e dessa forma cria-se um *loop*. Conseqüentemente, o pacote de ataque nunca vai deixar a rede. Portanto, se um atacante envia um grande número de pacotes ICMP `echo` com o endereço IP de origem forjado, ele pode consumir a banda passante e sobrecarregar a vítima. O serviço *chargen* também pode ser utilizado para amplificar o tráfego de ataque. Apenas com o envio de um pequeno pacote para tal serviço, uma grande seqüência de caracteres é gerada e enviada como resposta. Desta forma, supondo um fator de amplificação de 50, o tráfego enviado pelo atacante a 128 kbps chega na vítima a 6,4 Mbps.

Ataques a Infra-estrutura de Rede

Ataques de negação de serviço muitas vezes são direcionados a sítios famosos na Internet. Desta forma, os seus autores conseguem uma fama maior como resultado do ataque. Nestes casos, é provável que a própria candidata à vítima seja superdimensionada, ou seja, possua recursos de processamento e de memória em abundância. Como conseqüência, ataques distribuídos de pequena escala não conseguem consumir tais recursos rápido o suficiente para que a vítima negue o seu serviço para usuários legítimos. Como uma alternativa, o atacante pode tentar concentrar seus esforços em algum ponto

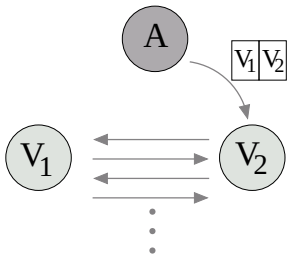

Figura 1.9. Ataque através da inundação UDP: o atacante A envia pacotes para a vítima V_2 se passando pela vítima V_1 para provocar uma troca infinita de mensagens entre as vítimas.

crucial para o funcionamento do serviço que não dependa da vítima. Um exemplo é tentar consumir toda a banda passante da vítima com o tráfego de ataque. Desta forma, por mais que a vítima consiga atender a todas as requisições que lhe chegam, muitas requisições são ainda perdidas na infra-estrutura de rede. Neste caso, o ponto onde as requisições são perdidas é algum roteador entre o atacante e a vítima onde existe um "gargalo," ou seja, onde o tráfego direcionado à vítima é maior do que o enlace de saída pode suportar. Desta forma, requisições legítimas ao serviço da vítima provavelmente são descartadas neste roteador, uma vez que precisam disputar o mesmo recurso com o tráfego enviado pelo atacante. Tais ataques são difíceis de serem combatidos, já que os pacotes não precisam ter nenhum padrão semelhante que possibilite filtrá-los.

Outro ataque que pode negar o serviço da vítima sem atacá-la diretamente é uma inundação aos seus servidores DNS. Estes servidores são responsáveis por traduzir nomes, usados pelas pessoas, em endereços IP, usados pelos computadores. Como geralmente as pessoas utilizam nomes para acessar um determinado servidor, um ataque ao serviço de resolução de nomes acaba por negar serviços de outros servidores. Um ataque deste tipo foi recentemente direcionado ao sítio da empresa Microsoft [Wagner, 2002]. Na época, seus servidores DNS estavam localizados em um mesmo segmento de rede e foi possível atacar a todos de uma vez através da inundação deste segmento. Como resultado, milhares de usuários ficaram sem acesso aos serviços providos pela empresa. Atualmente, seus servidores estão espalhados geograficamente e possuem enlaces redundantes de forma a minimizar os efeitos de um ataque semelhante.

Um outro alvo potencial para ser atacado são os próprios roteadores responsáveis por encaminhar os pacotes até a vítima. A função dos roteadores é simplesmente encaminhar cada pacote recebido, de acordo com o seu endereço de destino. Para isso, os roteadores difundem certas informações na rede de forma que cada roteador consiga construir uma tabela de roteamento, responsável por associar cada endereço de destino com uma interface de saída. Além da simples inundação, outros tipos de ataques podem ser direcionados a estes roteadores. Um atacante pode, por exemplo, encher a sua tabela de roteamento para dificultar a busca de endereços na tabela e atrasar muito o encaminhamento dos pacotes. Ou ainda, o roteador pode ser enganado e acabar encaminhando o tráfego legítimo para um local errado ao invés de entregá-lo para a vítima.

Ataques por Vulnerabilidade

Uma outra forma de negar os serviços providos pela vítima é deixá-la inoperante de alguma forma. Uma das maneiras de atingir este objetivo é explorar alguma vulnerabilidade na implementação da pilha de protocolos ou da própria aplicação da vítima. Um exemplo deste tipo de vulnerabilidade ocorreu, por exemplo, na implementação do protocolo TCP em sistemas operacionais Windows recentemente [Microsoft, 2002]. Para a sua ativação, o atacante precisava construir um segmento TCP com determinadas características e enviá-lo para a vítima. Ao receber este segmento, a vítima passava para um estado inesperado e o sistema operacional abortava, causando o congelamento total do seu processamento. O envio periódico destes segmentos poderia deixar a vítima inoperante por um bom tempo, dependendo da vontade do atacante.

Ataques de negação de serviço também já exploraram vulnerabilidades no próprio protocolo IP. Quando um determinado pacote é grande demais para ser transmitido sobre uma tecnologia de rede, o protocolo IP permite a quebra do pacote em fragmentos menores e o envio de cada fragmento separadamente. Para isso, cada fragmento possui um identificador, de forma que receptor consiga agregar os fragmentos de um mesmo pacote, e um número de seqüência, para determinar aonde no pacote original aquele fragmento se encontra. O receptor então recebe os fragmentos e os concatena de forma a remontar o pacote original. Assim, o ataque consistia no envio de diversos fragmentos IP pertencentes ao mesmo pacote com números de seqüência que se sobrepunham [CERT, 1997]. Desta forma, a vítima recebia uns poucos fragmentos intencionalmente mal formados e o protocolo entrava em um estado não previsto, resultando no seu congelamento ou reinicialização. Mesmo com a correção de tais vulnerabilidades, ataques usando fragmentos IP ainda são usados em ataques de negação de serviço que visam consumir os recursos da vítima. Tendo recebido um fragmento com um novo número identificador, a vítima armazena este fragmento por um determinado período até que todos os fragmentos restantes sejam recebidos ou até que um temporizador estoure. Além disso, o processamento de cada fragmento é necessário para determinar a sua posição dentro do pacote original. Desta forma, um atacante pode inundar a vítima com diversos fragmentos, cada um com um identificador diferente, de forma a esgotar a memória da vítima e aumentar o seu processamento. Dependendo da freqüência com que tais fragmentos são enviados para a vítima, a condição para a negação do seu serviço pode ser atingida.

Gont [Gont, 2004] descreve um novo tipo de ataque a conexões TCP já estabelecidas através do uso de somente um pacote ICMP. Uma das funções destes pacotes é relatar eventuais erros que ocorrem durante o roteamento de um pacote IP desde o emissor até o destinatário. O autor percebeu que é possível enviar um pacote ICMP forjado para uma das entidades que se comunicam, de forma a fingir que algum erro ocorreu durante a transmissão de um pacote. Desta forma, a conexão TCP é desfeita e uma nova conexão precisa ser estabelecida para o uso do serviço. O envio contínuo destes pacotes pode interromper o andamento de um serviço baseado no protocolo TCP. A identificação da origem de tais ataques que só utilizam um pacote para negar o serviço da vítima ainda é desafio para os sistemas de rastreamento atuais.

Ataques Distribuídos

Os ataques de negação de serviço distribuídos são geralmente usados em ataques por inundação, quando uma estação sozinha não é capaz de consumir algum recurso da vítima. Portanto, diversas estações precisam ser usadas para gerar o tráfego de ataque em direção à vítima e negar o seu serviço. Estas estações geralmente não pertencem ao atacante e são simplesmente computadores comprometidos por alguma falha de segurança. Uma vez tendo comprometido uma estação, o atacante apaga os rastros deixados pela invasão e instala um programa para comandá-la remotamente. Estes computadores ficam então sob o controle do atacante e, por isso, são chamados de agentes, escravos ou zumbis, denotando que são comandados por uma outra entidade.

Diferentes maneiras podem ser usadas pelo atacante para penetrar no sistema de outras estações. Geralmente, estas estações não possuem um sistema atualizado com as últimas versões dos programas usados. Como vulnerabilidades são encontradas freqüentemente, é possível encontrar uma série de estações com versões ultrapassadas que possuem vulnerabilidades de segurança. Algumas destas vulnerabilidades podem ser exploradas remotamente e liberar o acesso ao invasor. Com o acesso à estação invadida, é possível se aproveitar de outras vulnerabilidades locais para obter determinados privilégios de administrador e controlar totalmente à estação invadida. Outra possibilidade que vem sendo muito usada atualmente é obter a senha de uma determinada conta do sistema através de ataque de força bruta. O invasor tenta inúmeras possibilidades como senha para a conta em questão até conseguir penetrar na estação atacada.

Como os ataques de negação de serviço distribuídos podem ser compostos por centenas ou milhares de zumbis [Mirkovic et al., 2004], a invasão manual de cada zumbi individualmente se torna uma atividade cansativa para o atacante. Além disso, à medida que o número de zumbis aumenta, é muito difícil controlar cada zumbi sem nenhuma forma de automação. Por isso, algumas ferramentas foram criadas de forma a encontrar estações vulneráveis e invadí-las automaticamente. Outras ferramentas possibilitam a criação de redes hierárquicas para permitir o controle de um grande número de zumbis. Através dessas ferramentas, o atacante consegue comandar a todos os zumbis que iniciem um ataque a uma determinada vítima com um simples comando.

De forma a esconder a sua identidade, o atacante pode usar diversas estações intermediárias entre si mesmo e os zumbis. Estas estações são denominadas mestres e cada uma controla um determinado conjunto de zumbis. O atacante controla diretamente cada estação-mestre que, por sua vez, repassa os comandos do atacante para os zumbis. A Figura 1.10 ilustra uma rede composta por um atacante A, pelas estações-mestre M_1, M_2 e M_3 e pelos zumbis $Z_1, Z_2, Z_3, \ldots, Z_{12}$ para atacar uma vítima V. Durante a execução de um ataque, A comanda a M_1, M_2 e M_3 que iniciem um determinado ataque direcionado a V. As estações-mestre então repassam o comando para os zumbis sob seu controle e eles inundam a vítima. O ataque escolhido pelo atacante A e realizado por cada zumbi Z_i pode ser qualquer um daqueles descritos anteriormente, inclusive ataques por refletores. Além da vantagem de manter a identidade do atacante escondida, o uso destas redes hierárquicas é essencial para o controle de uma grande quantidade de zumbis.

Para se rastrear o atacante, é necessário primeiro identificar os zumbis para depois descobrir quem são os mestres e, por fim, chegar até o atacante. Logo, quanto mais cama-

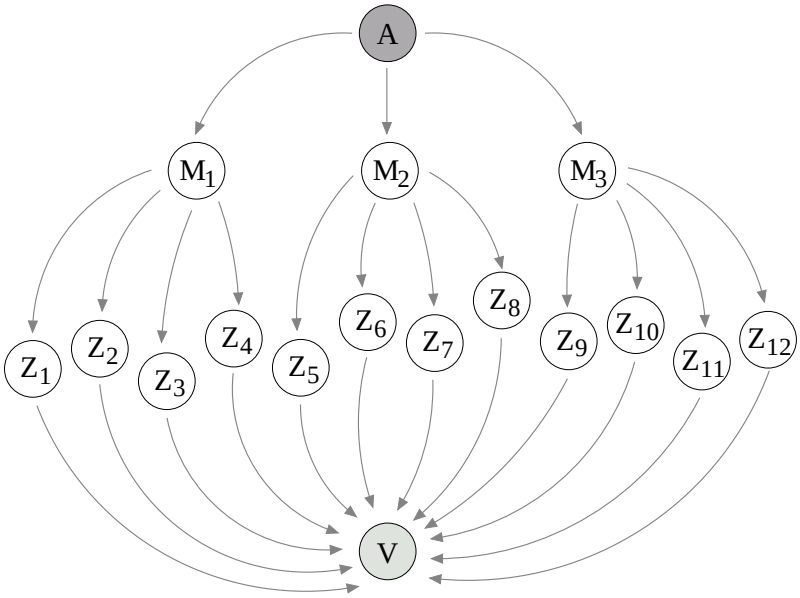

Figura 1.10. Ataque de negação de serviço distribuído.

das existem nesta hierarquia, mais protegido está o atacante. A Figura 1.11 ilustra duas técnicas utilizadas para o atacante comandar as estações-mestre. A primeira técnica usada por atacantes é entrar em diversas estações em seqüência antes de acessar os mestres. Inicialmente, o atacante *A* entra em uma estação intermediária I_1 e, por meio dela, o controle dos mestres é enfim realizado. Uma outra maneira de realizar este controle indireto é usar uma sala de bate-papo através do protocolo IRC (*Internet Relay Chat*), ou seja, um canal IRC. Neste caso, os mestres ou os próprios zumbis entram automaticamente em uma sala de bate-papo de um servidor escolhido pelo atacante e ficam esperando ordens. O atacante então entra na sala e envia os comandos para que o ataque seja iniciado [Gibson, 2001].

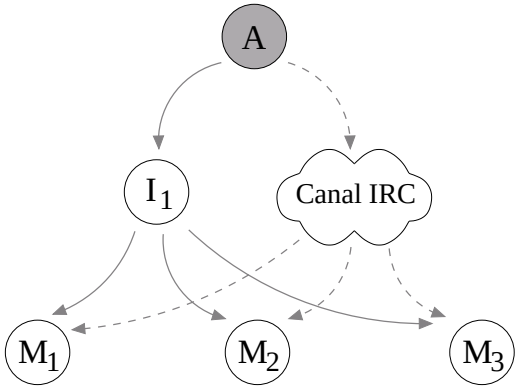

Figura 1.11. Estação intermediária ou um canal IRC para controlar as estações-mestre.

Diversas vantagens para o atacante surgem com a distribuição de ataques de negação de serviço. A primeira vantagem é conseguir deixar vítimas superdimensionadas inoperantes. Por possuírem recursos em abundância, estas vítimas são imunes a ataques de negação de serviço partindo de um único atacante. Entretanto, quando diversas estações são usadas para a geração de tráfego de ataque, a vítima pode ser seriamente afetada.

Um fato importante é que não importa o quão superdimensionada é uma determinada vítima, ela sempre pode ter o seu serviço negado desde que um número suficientemente grande de zumbis seja reunido para o ataque. Caso a vítima tente amenizar o efeito de um ataque distribuído através do aumento de seus recursos, o atacante pode simplesmente usar mais zumbis e obter o mesmo resultado.

Para interromper um ataque distribuído em andamento, a identificação de somente um zumbi não é suficiente. Como o tráfego agregado de diversos zumbis está inundando a vítima, é provável que, ao interromper um único zumbi, o efeito na vítima seja desprezível. Neste caso, é preciso identificar grande parte dos zumbis que estão atacando a vítima e tentar filtrar este tráfego de ataque o mais perto das fontes possível. Entretanto, só a identificação dos zumbis já é uma tarefa difícil que pode precisar da colaboração de diversos provedores de acesso. Além disso, é preciso contatar o responsável ou o administrador do zumbi para que a inundação originada por ele termine. À medida que o número de zumbis cresce, não é possível realizar esta tarefa para cada zumbi e é mais difícil interromper o ataque.

Como solução alternativa, é possível ainda tentar interromper a atividade de uma estação-mestre diretamente. Porém, é preciso antes identificar tais estações para depois tentar tomar alguma ação para interromper o ataque. Estas duas tarefas podem não ser simples. A identificação de uma estação-mestre exige primeiro a identificação de pelo menos um zumbi e a partir deste zumbi, tentar descobrir quem o está comandando remotamente. Além disso, as ferramentas de controle destas estações podem exigir algum tipo de autenticação e bloquear o acesso de quem não é autorizado. Desta forma, não é possível parar o ataque sem a intervenção do próprio administrador do zumbi.

1.3.5. Contramedidas aos Ataques de Negação de Serviço

Existem algumas medidas capazes de evitar ou, pelo menos, reduzir os danos causados por ataques de negação de serviço. Estas técnicas podem ser divididas em duas categorias: medidas preventivas e medidas reativas. As medidas preventivas buscam evitar ou reduzir a chance de que um ataque de negação de serviço ocorra. As medidas reativas, por outro lado, tentam lidar com ataques em andamento ou já encerrados. Para serem efetivas, os dois tipos de técnicas devem ser empregadas em conjunto. É importante utilizar medidas preventivas para reduzir o número de ataques; no entanto, é impossível garantir que não ocorrerão ataques. Para que as medidas preventivas sejam totalmente eficazes, é necessária a sua adoção em larga escala, possivelmente de forma cooperativa entre diversas administrações. Para estes ataques que escapam às medidas preventivas adotadas, resta a utilização de medidas reativas.

Medidas Preventivas

A medida preventiva mais imediata é a manutenção de todos os softwares dos sistemas finais devidamente atualizados. Este procedimento, além de minimizar os ataques por vulnerabilidades como os discutidos na Seção 1.3.4, também dificulta o domínio de possíveis zumbis. Fica claro, com isto, que a segurança dos usuários começa com os próprios usuários. Os responsáveis pelos sistemas finais não devem simplesmente espe-

rar que a infra-estrutura de rede forneça a segurança de seus sistemas, mas devem ser participantes ativos deste processo.

Uma técnica preventiva, denominada filtragem de ingresso (*ingress filtering*), foi sugerida para evitar que pacotes com endereços IP de origem forjados trafeguem na rede [Ferguson e Senie, 2000]. Implementada em alguns roteadores, a filtragem de ingresso descarta pacotes cujos endereços de origem não pertencem a determinados prefixos legitimamente anunciados. Em outras palavras, um roteador encarregado da agregação de rotas anunciadas por diversas redes em seu domínio deve impedir a saída de pacotes cujo endereço de origem não pertence a nenhuma dessas redes. A Figura 1.12 ilustra este conceito. Na figura, o atacante *A* está querendo atacar a vítima *V* usando pacotes com endereços de origem forjados, de forma a manter o anonimato. Seus pacotes seguem até a vítima *V* através dos roteadores R_1, R_2 e R_3. Caso a filtragem de ingresso esteja implementada em R_1, somente aqueles pacotes cujo endereço de origem pertence a faixa de endereços da Rede 1 são roteados. Pacotes com endereços de origem forjados são automaticamente impedidos de sair da rede e descartados por R_1. Desta forma, somente pacotes cujo endereço de origem estão dentro da faixa de endereços da Rede 1 podem ser usados para o ataque. Neste caso, a vítima *V* consegue descobrir facilmente a origem dos pacotes a partir de uma rápida análise no tráfego de ataque recebido. Roteadores podem ainda agregar faixas de endereços de origem para diminuir o processamento realizado por pacote. No exemplo, supondo que as faixas de endereços das Redes 1, 2, 3 e 4 podem ser agregadas em somente uma faixa de endereços, R_3 pode evitar a saída de pacotes forjados verificando somente se o endereço do pacote está dentro da faixa agregada.

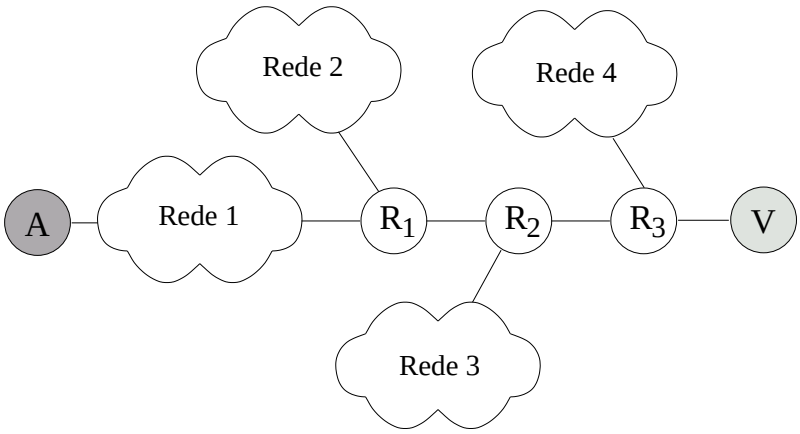

Figura 1.12. Filtragem de ingresso.

Apesar de simples, esta técnica apresenta desvantagens. Há uma indesejável dependência entre a segurança do destinatário final e as políticas adotadas nos roteadores ao longo do caminho. Na verdade, não existem incentivos para que domínios adotem a filtragem de ingresso como parte da política de segurança, uma vez que somente estações fora do próprio domínio são beneficiadas. Além disso, a filtragem de ingresso precisa ser implementada em uma escala global para ter efeito. Caso a implantação seja restrita a poucos roteadores, somente ataques passando por esses roteadores serão controlados. Uma outra desvantagem é processamento adicional inserido durante o roteamento. Como a filtragem influencia diretamente no processo de roteamento, o exame do endereço de

origem de cada pacote pode requisitar certos recursos que nem todo roteador disponibiliza. Existem ainda tecnologias que podem ser afetadas pela filtragem de ingresso, como o IP Móvel [Perkins, 2002], pois utilizam legitimamente pacotes com endereços de origem forjados.

Uma generalização da técnica de filtragem de ingresso foi proposta por Li *et al.* [Li et al., 2002]. Os autores propõem um protocolo que fornece aos roteadores algumas informações que possibilitam validar o endereço de origem dos pacotes. A idéia básica do protocolo é que mensagens contendo informações sobre endereços de origem válidos sejam propagadas a partir da própria rede de origem para todos os roteadores da rede. Desta forma, cada roteador consegue construir uma tabela de entrada que associa cada interface a um conjunto de endereços de origem válidos. Ao receber um pacote por uma de suas interfaces, o roteador verifica se o endereço de origem do pacote está dentro do conjunto de endereços associados àquela interface. Caso o endereço esteja dentro do conjunto, o pacote é roteado normalmente e, em caso contrário, ele é descartado. O protocolo proposto é semelhante a um protocolo de roteamento, onde as informações sobre os endereços de destinos são propagadas de forma que os roteadores possam construir uma tabela associando endereços de destino com interfaces de saída, a chamada tabela de roteamento. A diferença é que protocolos de roteamento visam determinar por qual interface um pacote será encaminhado enquanto que o protocolo de validação de endereço de origem verifica se o pacote foi recebido pela interface adequada.

A desvantagem principal do protocolo de validação proposto é o processamento adicional necessário durante o roteamento de cada pacote. Caso este protocolo venha a ser implementado, roteadores terão que fazer consultas a cada uma das tabelas. Uma consulta na tabela de entrada é necessária para determinar se o endereço de origem é válido e outra consulta na tabela de roteamento é necessária para determinar a interface de saída e o próximo nó a receber o pacote. Além disso, o protocolo proposto também apresenta os problemas de um protocolo de roteamento convencional, como a necessidade de atualizações periódicas, a reação às mudanças topológicas da rede, a garantia da integridade e da autenticidade das mensagens trocadas e o uso eficiente da banda. Como objetivo final, ele deve ainda garantir que pacotes com endereços de origem forjados sejam descartados e que nenhum pacote com endereço legítimo o seja. Tudo isso torna o protocolo complexo.

Medidas Reativas

Nos casos em que o ataque não pode ser evitado, não resta alternativa se não tentar fazê-lo parar e buscar localizar o atacante para que sejam tomadas medidas reparativas. Para isto, o primeiro passo é necessariamente obter informações sobre a origem do ataque.

O rastreamento de pacotes surge como uma forma de identificar a rota percorrida por um determinado pacote até o seu verdadeiro emissor. No caso de ataques de negação de serviço, a identificação dos atacantes é importante tanto para interromper o ataque através de filtragem como para a adoção de medidas judiciais contra o próprio atacante. Entretanto, o rastreamento é ainda mais geral e pode ser útil em qualquer ocasião onde é necessário determinar a origem ou a rota percorrida por um determinado pacote.

A importância do rastreamento é ainda maior quando se leva em consideração que,

devido à arquitetura da Internet, não se pode confiar no endereço de origem de nenhum pacote como forma de identificação de sua verdadeira origem. É impossível apenas analisando o pacote determinar se o endereço é verdadeiro ou foi forjado. Desta forma, para ser confiável, toda informação sobre a origem de um ataque deve ser obtida através do rastreamento, pois se deve sempre levar em conta a possibilidade de endereços forjados.

1.4. O Rastreamento de Pacotes IP

O rastreamento de pacotes tem como finalidade identificar a rota percorrida por um determinado pacote e o seu verdadeiro emissor. No caso de ataques de negação de serviço, é fundamental identificar a origem do ataque, mesmo que estes não empreguem endereços de origem forjados, de forma a filtrar o tráfego de ataque na sua origem. Em ataques por refletor e através de zumbis, o rastreamento deve ser realizado para se identificar o refletor ou zumbi e a partir destes, um novo rastreamento deve ser realizado para se chegar aos verdadeiros atacantes.

As técnicas de rastreamento de pacotes IP podem ser divididas em duas categorias: sistemas de rastreamento sem estado e sistemas de rastreamento baseados em auditorias [Laufer et al., 2005b]. Os sistemas de rastreamento sem estados não armazenam nenhum tipo de informação sobre a origem dos pacotes nem na infra-estrutura da rede, nem nas estações finais. Neste caso, toda a atividade de rastreamento se baseia no estado da rede no momento em que o rastreamento é feito. Por outro lado, os sistemas baseados em auditoria armazenam de alguma forma, durante o processo de envio dos dados, informações que podem ser úteis para reconstruir o caminho real percorrido pelos pacotes. A Seção 1.4.1 discute possíveis soluções de rastreamento sem o armazenamento de informações e a Seção 1.4.2 trata das soluções que se baseiam na auditoria de dados armazenados durante o processo de envio dos pacotes.

1.4.1. Sistemas de Rastreamento Sem Estado

As soluções de rastreamento sem estados possuem a vantagem de não exigir uma estrutura capaz de armazenar informações sobre os pacotes enviados em tempo real. A busca por uma técnica que não requeira armazenagem de dados é significativa, pois o número de pacotes comutados na rede é enorme e, como conseqüência, o volume de dados a ser armazenado é grande. Também existe o aspecto legal que não permite que se armazene conteúdo de pacotes que circulam na rede. No entanto, sem o auxílio de dados de auditoria, a única opção para determinar a rota seguida pelos pacotes do ataque é testar os enlaces (interfaces) da rede para determinar se o tráfego de ataque passa ou não por esse enlace. Desta forma, estas soluções só podem ser utilizadas na detecção de ataques por inundação e enquanto os ataques ainda estão em andamento, sendo incapazes de realizar o rastreamento *post mortem*.

A solução atualmente empregada para o rastreamento de ataques é mais precária possível, pois se serve de procedimentos que requerem a intervenção humana. Assim, quando ocorre um ataque, a vítima entra em contato com o administrador da rede do seu provedor de serviço (*Internet Service Provider* - ISP) e solicita a determinação da rota de ataque. A vítima ou o administrador do ISP procura uma característica (assinatura) dos pacotes usados no ataque ou faz-se uma análise do tráfego no roteador associado à vítima.

É feito um teste no roteador mais próximo da vítima para determinar por qual enlace (interface de rede) o tráfego do ataque chega a esse último roteador. Uma vez determinado o enlace correto, o procedimento é repetido no roteador localizado na outra extremidade do enlace. Depois, salto-a-salto, este processo é repetido até que seja determinado o roteador de onde parte o ataque. Pode ser necessário testar até d roteadores, onde d é o diâmetro da rede. Em alguns casos, é necessária a interação entre provedores de serviço, o que torna o processo lento.

Em situações nas quais é possível manipular o roteamento da rede através da inclusão de algumas rotas estáticas, uma opção é fazer com que todo o tráfego destinado à vítima seja direcionado a um roteador previamente determinado. Se esse roteador for configurado de forma a estar no centro da rede, pode-se iniciar um teste de enlaces salto-a-salto partindo do centro da rede. Desta forma, o teste de enlaces é feito em no máximo $d/2$ roteadores. Se a análise estiver sendo feita numa rede similar a um ISP, na qual parte-se do princípio de que os ataques só podem vir de alguém conectado a um dos roteadores de borda, existem duas outras opções para o teste de enlaces. A primeira, caso a rede disponha de recursos para a construção de topologias lógicas sobre a topologia física, é a construção de uma rede lógica de rastreamento em sobreposição (*overlay*) ligada a todos os roteadores de borda da rede em questão. Com isso, é possível realizar o teste de enlaces salto-a-salto partindo da vítima até o roteador de borda por onde o tráfego de ataque entra na rede através da rede lógica de rastreamento. Neste caso, o número máximo de roteadores testados é função do diâmetro da rede lógica. A outra opção, caso deseje-se apenas determinar por qual roteador o tráfego de ataque entra na rede, é analisar estatísticas de tráfego nos roteadores de borda da rede. Assim, baseado na assinatura do ataque, é possível determinar qual roteador de borda é o responsável pela entrada do tráfego de ataque na rede.

As duas técnicas de rastreamento sem estados apresentadas nesta seção são, na verdade, formas de automatizar e otimizar o teste dos enlaces. A primeira técnica, chamada *CenterTrack* [Stone, 2000], tem o escopo limitado à rede de um único ISP e busca determinar por onde o tráfego de ataque entra na rede deste ISP. Neste sistema, o rastreamento é otimizado através da utilização de uma rede lógica de rastreamento em sobreposição, na qual são realizados os testes de enlace salto-a-salto. A segunda técnica é conhecida como “inundação controlada” [Burch e Cheswick, 2000] e tem como objetivo automatizar os testes de enlace salto-a-salto numa rede composta por diferentes ISPs e, portanto, sem uma administração central.

Para que a técnica *CenterTrack* possa ser utilizada, é necessária a construção de uma rede lógica de rastreamento na rede do ISP em questão. Desta forma, esta técnica aumenta a complexidade da rede e acarreta um aumento nas tarefas administrativas do ISP. Cuidados especiais devem ser tomados, por exemplo, para evitar que os túneis utilizados na construção da topologia lógica da rede de rastreamento interfiram na operação dos protocolos de roteamento que estejam sendo utilizados no interior da rede. No entanto, esta técnica tem como vantagem requerer a presença de ferramentas de diagnóstico apenas nos nós de borda e nos nós que compõem o *backbone* da rede de rastreamento.

O *backbone* da rede de rastreamento pode ter diversas arquiteturas, cada uma adequada a certas características da rede física do ISP [Stone, 2000], mas deve estar ligada

a todos os roteadores de borda da rede. Desta forma, este *backbone* liga todas possíveis entradas do tráfego de ataque a todas as possíveis vítimas. Uma vez descoberto o ataque, são criadas rotas estáticas apontando para a vítima que farão com que após a convergência do protocolo de roteamento todo o tráfego direcionado à vítima passe pela rede de rastreamento.

Após a reconstrução das rotas através da rede lógica de rastreamento, pode-se realizar testes de enlace salto-a-salto, partindo da vítima, através desta rede lógica. Embora estes testes não identifiquem com exatidão *todos* os roteadores da rota, ele resultará na descoberta do roteador por onde o tráfego entra na rede. Caso deseje-se filtrar o tráfego de ataque, esta filtragem deve ser realizada neste roteador de entrada.

Esta técnica apresenta algumas limitações. Uma destas limitações que merece destaque é que o sistema não ajuda no rastreamento de ataques originados no *backbone* do ISP, pois este roteador pode não fazer parte da rede de rastreamento. Outro problema surge quando a vítima é um dos roteadores do *backbone* da rede do ISP. Neste caso, o procedimento de direcionar o tráfego destinado à vítima através da rede de rastreamento pode resultar em *loops* e colapso de túneis. A utilização de túneis gera, ainda, mais dois problemas. O primeiro deles é a sobrecarga introduzida pelo uso de túneis. Desta forma, se o ataque for realizado com vários pacotes pequenos, o uso de túneis podem aumentar o tamanho do pacote significativamente. Em última instância, os túneis servirão como um amplificador do ataque. Além disso, se os túneis não forem autenticados de alguma forma, um ataque mais inteligente pode atrapalhar a operação do sistema forjando pacotes de modo a que estes pacotes sejam injetados diretamente nos túneis. Uma questão que também merece atenção é que este sistema redireciona o tráfego endereçado para a vítima. Esta característica possibilita ao atacante descobrir através de ferramentas como o *traceroute* que alguma coisa está errada e que ele está possivelmente sendo rastreado. Sabendo disso, o atacante pode parar o ataque antes que o processo de identificação esteja completo. Deve-se destacar, por fim, que esta técnica pode ser facilmente adaptada para se tornar uma técnica baseada em auditoria. Para isto, é necessário que equipamentos capazes de "farejar" a rede sejam incluídos no *backbone* da rede de rastreamento.

A técnica da inundação controlada busca automatizar o processo de teste de enlaces em redes compostas por diferentes ISPs. Esta técnica necessita que algumas ferramentas estejam disponíveis em todos ou, pelo menos, na maior parte dos roteadores da rede considerada. A principal vantagem deste sistema é possibilitar o rastreamento sem a intervenção dos ISPs envolvidos. Na realidade, esta técnica pode até mesmo ser empregada sem que a administração do ISP tome conhecimento.

Este sistema se baseia no teste de enlace salto-a-salto na rede inteira. Para isso, é necessário que o responsável pelo rastreamento tenha conhecimento da topologia da rede como um todo. Isto pode ser obtido através de *traceroutes* da vítima para as outras redes. Este processo, além de custoso do ponto de vista de tempo e recursos, é complicado caso existam enlaces assimétricos. Embora se possa assumir que a maior parte das rotas são simétricas, pode ser necessária a utilização de outros meios para obter uma visão suficientemente detalhada da rede.

Um ponto interessante da inundação controlada é a forma com que o teste de enlace é realizado. Ao invés de analisar o tráfego que passa por um determinado en-

lace e comparar com uma assinatura conhecida do ataque, na inundação controlada o teste de enlace é feito através da inundação do enlace em questão. Após esta inundação, determina-se se o enlace pertence ou não à rota de ataque observando o fluxo do ataque que chega à vítima. Se o fluxo do ataque diminui após a inundação do enlace, assume-se que o enlace faz parte da rota de ataque e a diminuição do fluxo na vítima se deu por descarte de parte dos pacotes de ataque devido ao congestionamento criado no enlace testado. Pode-se dizer que neste sistema o teste de enlace é realizado de forma indireta. Duas questões importantes surgem com este tipo de teste de enlace. A primeira é a influência do comportamento do fluxo do ataque sobre a identificação dos enlaces pertencentes à rota de ataque. Uma vez que a identificação depende da observação de variações do fluxo na vítima, se o ataque for originalmente feito com um fluxo imprevisível de pacotes, o teste de enlaces se torna muito difícil de ser realizado com alguma confiança. Além disso, é necessário que o responsável pelo rastreamento seja capaz de efetivamente inundar o enlace testado, o que se torna mais difícil à medida que a capacidade do enlace aumenta. É importante destacar que essa inundação deve ser idealmente contida ao enlace em questão para não mascarar resultados nem congestionar outras partes da rede. Stone [Stone, 2000] sugere o uso do serviço de geração de caracteres (*chargen*). Esta ferramenta gera um fluxo contínuo de dados partindo do roteador em direção a quem se conecta ao serviço. A inundação pode ser restrita ao enlace em questão se o responsável pelo rastreamento utilizar o serviço forjando o endereço de origem do pedido para fazer parecer que quem está se conectando é o roteador na outra extremidade do enlace testado. Os testes realizados demonstram que na maioria dos casos esta ferramenta é capaz de inundar o enlace, especialmente considerando-se que esta inundação é facilitada pelo tráfego legítimo que circula pela rede. Os resultados mostram que na maioria das vezes é possível identificar o segmento da rede de onde parte o ataque. Entretanto, em alguns casos, o teste de enlace pode apresentar resultados que ajudam pouco na determinação da rota de ataque.

Um aspecto importante da inundação controlada é a sua implicação ética. Inundar um enlace para testar se ele faz parte da rota de um ataque de negação de serviço é, de certa forma, realizar um ataque de negação de serviço ao ataque original. Este segundo ataque, ao inundar a rede, pode prejudicar outros usuários. Surge então a pergunta se o rastreamento não pode causar mais dano do que o próprio ataque. Espera-se que se os testes de enlace são feitos na escala temporal correta (apenas poucos segundos), a maior parte dos usuários sequer percebem mudanças na rede. Por fim, há a discussão sobre a utilização de serviços disponíveis nos roteadores, como o *chargen*. O fato de um ISP deixar um serviço funcionando em um roteador não significa que este ISP concorda que qualquer um se utilize deste serviço.

1.4.2. Sistemas de Rastreamento Baseados em Auditoria

O rastreamento de pacotes baseado em auditoria consiste basicamente na coleta de informações sobre os pacotes que circulam na rede a fim de viabilizar a reconstrução do caminho percorrido por pacotes provindos de atacantes. Este tipo de sistema pode ser classificado segundo o local onde as informações coletadas são armazenadas. Assim, as informações podem ser armazenadas nas estações finais, na infra-estrutura de rede ou ainda em ambas.

Armazenamento nas Estações Finais

O mecanismo mais simples de rastreamento de pacotes baseado em auditoria é a inclusão do endereço IP de cada roteador pelo qual o pacote passa. Este é um esquema similar à opção de Record Route do IP [Postel, 1981], na qual todo pacote possui o caminho inteiro percorrido na rede, da fonte ao destino. Além da simplicidade, este esquema permite o rastreamento do caminho de ataque com apenas um pacote, o que seria ideal em ataques distribuídos. No entanto, ele apresenta algumas desvantagens que o torna praticamente inviável [Savage et al., 2001]. A adição de dados ao pacote durante o roteamento implica um acréscimo significativo de processamento. Outro problema é o aumento do tamanho do pacote a cada roteador, podendo acarretar em fragmentações desnecessárias que podem sobrecarregar os roteadores.

Uma característica bastante comum nos sistemas com armazenamento nas estações finais é a coleta de informações através da marcação de pacotes. A marcação de pacotes consiste na utilização de campos no interior do pacote que possibilitem registrar uma determinada característica. Para efeito de rastreamento, são visadas as informações sobre o caminho percorrido pelo pacote do nó origem até o nó destino. Para não se alterar o tamanho original do pacote é comum usar campos pouco usados do cabeçalho IP para transportar estas informações. A marcação pode ser determinística, quando todos os pacotes são marcados, ou probabilística, quando os pacotes são marcados aleatoriamente segundo uma determinada probabilidade.

A marcação de pacotes probabilística (*Probabilistic Packet Marking* - PPM) foi introduzida originalmente por Savage *et al.* [Savage et al., 2001]. Nesta abordagem, cada roteador probabilisticamente insere informações sobre si mesmo no cabeçalho IP do pacote. Após o recebimento de uma certa quantidade de pacotes, a vítima é capaz de reconstruir a rota de ataque. A Figura 1.13 mostra a idéia básica do mecanismo de marcação de pacotes.

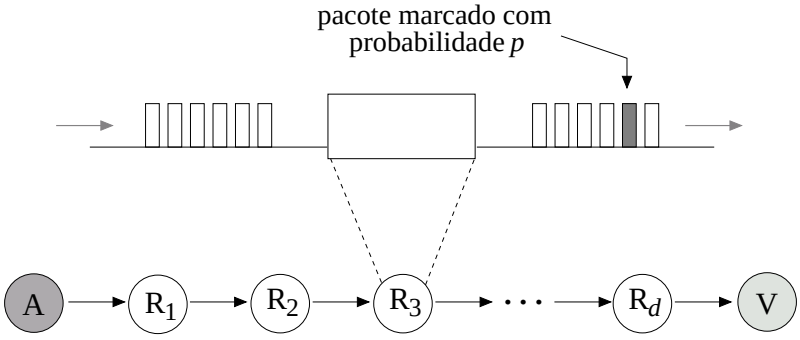

Figura 1.13. A marcação probabilística de pacotes.

O atacante A envia pacotes em direção da vítima V. Estes pacotes seguirão o caminho indicado, passando pelos roteadores $R_1, R_2, R_3, \ldots, R_d$. Cada roteador do caminho irá acrescentar ao cabeçalho do pacote o seu endereço IP, com uma certa probabilidade p. A vítima V deve armazenar um número suficiente de pacotes de modo a garantir a existência de pelo menos um pacote com o endereço de cada roteador do caminho. Deste modo,

após o recebimento dos pacotes, a vítima é capaz de reconstruir o caminho até o atacante. A ordem dos roteadores na rota reconstruída é deduzida a partir da quantidade de pacotes marcados por cada roteador. A probabilidade de um pacote chegar marcado por um determinado roteador é inversamente proporcional à sua distância para a vítima. Isto porque quanto mais próximo da vítima, menor é a probabilidade de um outro roteador sobrescrever o endereço já marcado. Assim, quanto maior é a quantidade de pacotes marcados por um determinado roteador, menor é a sua distância para a vítima. A probabilidade α_i de um pacote chegar marcado por um determinado roteador R_i é expressa por

$$\alpha_i = p(1-p)^{d-i}, \tag{1}$$

onde p é a probabilidade de um pacote ser marcado por R_i, d é o número de roteadores na rota de ataque e $(d-i)$ é a distância do roteador R_i à vítima.

Um dos problemas da marcação de pacote é a quantidade de pacotes que a vítima deve coletar. Por exemplo, se a distância do atacante for igual a 15 ($d = 15$) e $p = 0,51$ a vítima deve armazenar em média 42 mil pacotes para receber pelo menos um pacote do último roteador. Sendo que, para obter uma garantia de 95%, o número de pacotes armazenados passa para aproximadamente 300 mil pacotes. Por isso, todos os mecanismos baseados na marcação de pacotes partem da premissa que os ataques de negação de serviço são caracterizados pelo envio de milhares de pacotes [Savage et al., 2001]. Esta grande quantidade de pacotes pode tornar a reconstrução da rota um procedimento lento. Outro problema é a falta de robustez em relação a múltiplos atacantes. Neste caso, haverá mais de um roteador com a mesma distância dificultando a definição da rota de ataque. A única alternativa possível seria o conhecimento da topologia para distinguir entre os possíveis caminhos. Por fim, o atacante pode ainda tentar forjar um sufixo da rota ao acrescentar falsos endereços aos pacotes de ataque. A probabilidade α_0 de um pacote chegar a vítima com o endereço forjado pelo atacante é

$$\alpha_0 = (1-p)^d. \tag{2}$$

Existe um compromisso entre a capacidade do atacante de forjar endereços e a quantidade de pacotes necessários para a reconstrução da rota. Quanto maior é o valor de p, menor é a probabilidade da vítima receber um pacote marcado somente pelo atacante, conforme a Equação 2. Por outro lado, quanto menor é o valor de p, menos pacotes recebidos são necessários para o rastreamento. Existe um valor ótimo para p que minimiza a quantidade de pacotes a serem coletados. Um valor abaixo ou acima deste valor implica um aumento da quantidade de pacotes.

Apesar das desvantagens, a marcação de pacotes apresenta algumas vantagens interessantes. A marcação de pacotes não exige uma cooperação entre os ISPs. Diferentemente dos mecanismos baseados em inundação, o tráfego adicional acrescentado pelo esquema de marcação de pacotes não é significativo. Além disso, um mecanismo baseado na marcação de pacotes permite a descoberta da rota de ataque mesmo após o seu término. Por fim, a marcação de pacotes não representa uma grande sobrecarga para os roteadores.

Este é o princípio básico do mecanismo de rastreamento baseado na marcação de pacotes. Muitos trabalhos propuseram pequenas adaptações a fim de minimizar as desvantagens deste novo paradigma. Savage *et al.* propõem a amostragem de enlaces (arestas)

no lugar de amostragem de nós (vértices). Uma aresta pode representar um enlace entre dois roteadores conectados diretamente, ou através de outros roteadores que não implementam o mecanismo. Assim, cada pacote é marcado não apenas por um determinado nó, mas por dois nós consecutivos. Esta pequena modificação no esquema de marcação resolve o problema dos múltiplos atacantes, além de facilitar a implementação progressiva. Para tanto, são utilizados dois campos no cabeçalho do pacote onde serão armazenados os endereços de começo e fim da aresta e um campo para computar a distância, como mostra a Figura 1.14.

Figura 1.14. Espaço necessário nos mecanismos de amostragem de arestas.

Quando um nó decide marcar um determinado pacote, ele deve acrescentar seu endereço no campo *endereço 1* e em seguida zerar o campo *distância*. Caso contrário, o nó deve checar o campo *distância*. Sempre que o valor deste campo for igual à zero, o nó deve inserir seu endereço ao campo *endereço 2* e incrementar o valor do campo *distância*. Para qualquer outro valor, o roteador deve apenas incrementar o valor do campo *distância*. Ao final da coleta dos pacotes de ataque, a vítima terá informações sobre diferentes arestas, com diversas distâncias da vítima, a partir das quais é possível construir um grafo, onde a vítima é a raiz e os roteadores mais próximos dos atacantes são as folhas. O número médio X de pacotes necessários para a reconstrução da rota pode ser calculado segundo a expressão do problema do coletador de cupons. Nesse problema, é calculado o número médio de cupons que devem ser tirados de uma urna de forma a se ter pelo menos um cupom de cada tipo. Assim, o número médio X de pacotes necessários é expresso por

$$E(X) \leq \frac{\ln(d)}{p(1-p)^{d-1}}, \tag{3}$$

onde d representa a distância entre o atacante e a vítima e p a probabilidade de marcação. Este valor é referente a uma rota de ataque. No caso da amostragem de arestas, mesmo que o atacante consiga forjar um endereço, o efeito será o acréscimo de uma aresta ao final da rota. Isto significa que o atacante é incapaz de forjar arestas entre a vítima e o último roteador da rota de ataque.

O fato de utilizar um espaço adicional no campo do cabeçalho IP cria um problema de incompatibilidade com as versões anteriores. No mecanismo de amostragem de arestas, é necessário um espaço de 72 bits (Figura 1.14). Para solucionar este problema, Savage *et al.* propõem uma adaptação ao mecanismo de amostragem de arestas com o objetivo de diminuir o tamanho do espaço necessário no cabeçalho IP, de maneira a utilizar um espaço já existente e raramente utilizado. Assim, é proposta a utilização do campo de identificação de fragmentação (*IP identification*), que contém 16 bits. A novidade está na codificação da identificação das arestas, que antes era feito através dos endereços de começo e fim das arestas. Nesta nova versão, os endereços são intercalados pelo seu próprio *hash* e divididos em k fragmentos, como mostra a Figura 1.15. Assim, ao marcar um

pacote, ao invés de acrescentar seu endereço, o roteador deve acrescentar apenas um desses fragmentos, escolhido aleatoriamente, e escrever em um outro espaço do cabeçalho o *identificador de fragmento* (*ID frag*), além de zerar o campo *distância*. O nó seguinte, caso não marque o pacote, após verificar que o campo *distância* está em zero, deve escrever no campo *fragmento* o resultado da operação OU exclusivo (XOR) entre o *n*-ésimo fragmento de seu endereço com o fragmento já escrito, onde *n* é o número indicado no campo *identificador de fragmento* (Figura 1.16).

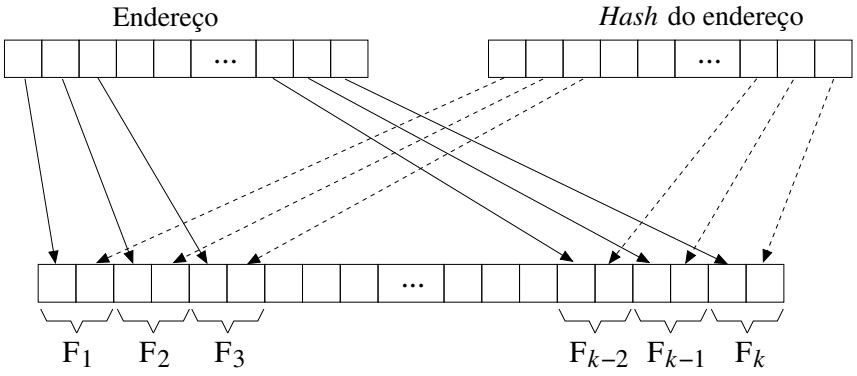

Figura 1.15. Geração dos fragmentos.

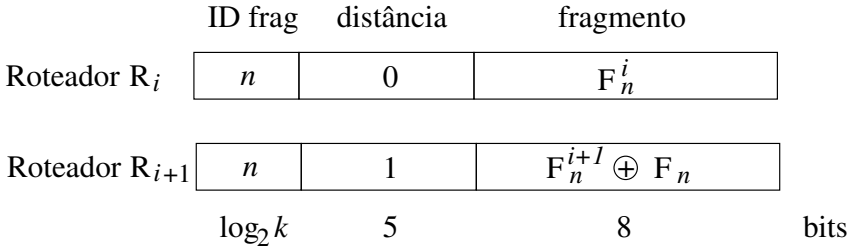

Figura 1.16. Marcação do pacote na amostragem de arestas, usando fragmentos.

Apesar de permitir a utilização de apenas 16 bits e manter a compatibilidade com as versões anteriores, este novo esquema aumenta o número de pacotes necessários para a reconstrução da rota de ataque, pois é necessário coletar pelo menos uma amostra de todos os fragmentos de cada aresta. Isto multiplica por *k* o número de pacotes necessários. Além disso, em um ataque distribuído com *m* atacantes, mesmo de posse de pelo menos uma amostra de cada um dos *k* fragmentos de cada aresta, no final, a vítima terá *m* exemplares diferentes com a mesma distância e o mesmo identificador de fragmento, provindas de *m* roteadores diferentes. Assim, para remontar todos os fragmentos de um mesmo roteador e obter seu endereço (intercalado pelo *hash*), a vítima deve testar todas as combinações possíveis e checar o *hash* de cada uma. Dependendo do número de atacantes, este procedimento pode implicar um processamento significativo. Mesmo tendo realizado todo este procedimento, ainda existe a possibilidade de obter endereços que, apesar de possuir um *hash* válido, não representam enlaces existentes. Estes enlaces são chamados de falsos positivos.

Pode-se mostrar que a proposta de amostragem de fragmentos sugerida por Savage *et al.* requer uma grande carga computacional e produz um significativo número de falsos positivos no processo de reconstrução da rota de ataque, na presença de múltiplos atacantes [Song e Perrig, 2001]. Por exemplo, com somente 25 atacantes, esta abordagem pode levar dias para computar a rota de ataque, apresentando milhares de falsos positivos. Assim, Song e Perrig [Song e Perrig, 2001] propõem um novo mecanismo de rastreamento de pacotes baseado na amostragem de arestas, que apresenta uma complexidade e número de falsos positivo inferior à proposição de Savage *et al*. A principal diferença entre as duas propostas está na codificação das arestas. Ao invés de acrescentar um fragmento do endereço IP intercalado com o *hash* do endereço, é acrescentado o valor do *hash* do endereço IP. Para isto, é definida a utilização de duas funções *hash* h e h', ambas de 11 bits. Cinco bits são utilizados para o campo *distância*, totalizando 16 bits. Desta forma, antes de encaminhar um pacote ao próximo nó, o roteador R_i deve decidir se irá marcar o pacote, com uma probabilidade p. Sempre que um pacote é marcado, o roteador insere o valor $h(E_i)$ no campo *aresta*, onde E_i é o seu endereço, e zera o campo *distância*. Caso o pacote não seja marcado, o roteador R_i deve checar o campo *distância*. Se o valor for igual à zero, o roteador deve atualizar o conteúdo do campo *aresta* com o resultado da operação OU exclusivo (XOR) entre o conteúdo do campo *aresta* e o valor $h'(E_i)$, e incrementar o campo *distância*. Para qualquer outro valor, apenas o campo *distância* é incrementado, como mostra a Figura 1.17.

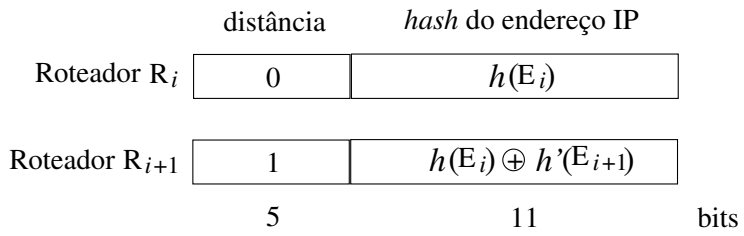

Figura 1.17. Marcação do cabeçalho segundo Song e Perrig.

A reconstrução é bastante simples. Primeiramente, considera-se que a vítima contém um mapa dos roteadores da Internet, representado por um grafo cuja raiz é a própria vítima. Assim, após a coleta de pacotes, a vítima deve comparar o valor $h(F_v)$ com o campo *aresta* de todos os pacotes cujo campo *distância* tem o valor zero, onde F_v contém todos os filhos diretos da vítima. Todos os endereços cujo resultado da comparação é positivo são acrescentados a um conjunto denominado S_0. O conjunto S_i contém todos os endereços IP cujo resultado da comparação é positiva, onde i representa a distância para a vítima. Após a identificação dos roteadores de distância zero, a vítima realiza a operação de OU exclusivo (XOR) entre o valor $h'(S_0)$ e o campo *aresta* de todos os pacotes cujo campo *distância* é igual a um, e comparar o resultado com $h(F_0)$, onde F_0 são todos os filhos do conjunto S_0. Neste novo esquema, um falso positivo pode ocorrer quando o resultado da função *hash* de dois nós irmãos, filhos do mesmo roteador, tem o mesmo valor. A probabilidade deste evento ocorrer é 2^{-11}. No caso da existência de muitos vizinhos a probabilidade da ocorrência de falso positivo aumenta significativamente. Para resolver este problema, é proposta uma pequena alteração no procedimento de marcação. Ao invés

de utilizar apenas duas funções *hash* independentes, são utilizado dois conjuntos g e g' de funções *hash*, todas independentes. Ao marcar um pacote, o roteador em questão deve escolher uma função *hash* do grupo g a ser utilizada. Após fazer a marcação e zerar o campo *distância*, ele deve inserir o valor sorteado no campo *identificador de função hash (ID hash)*. O roteador seguinte, que fará a operação OU exclusivo (XOR) para delimitar a aresta, deve utilizar a função *hash* com o mesmo ID *hash*, mas do grupo g'. Durante a reconstrução, uma determinada aresta só é considerada como verdadeira se possuir um número mínimo de marcações, cada uma com um valor *hash* diferente. Desta forma, os falsos positivos diminuem. Song e Perrig ainda propõem um mecanismo de marcação de pacotes com autenticação, baseado no esquema anterior, a fim de evitar que roteadores comprometidos possam forjar arestas e dificultar ou até mesmo impedir que a rota de ataque seja construída.

Neste sistema de marcação de pacotes, pode-se calcular a capacidade do atacante de forjar uma aresta [Park e Lee, 2001]. Para isto, basta que o atacante faça uma marcação e o pacote não seja marcado por nenhum roteador do caminho. A probabilidade de um pacote chegar na vítima sem ter sido marcado por nenhum roteador é dada pela Equação 2. Para que a vítima receba mais pacotes marcados pelo atacante que pacotes marcados pelo primeiro roteador do caminho, basta que:

$$\begin{aligned} \alpha_0 &\geq \alpha_1 \\ (1-p)^d &\geq p(1-p)^{d-1} \\ 1-p &\geq p \\ p &\leq 0,5. \end{aligned} \tag{4}$$

Para que a vítima receba mais pacotes marcados pelo atacante que por todos os outros roteadores, basta que:

$$\begin{aligned} \alpha_0 &\geq \sum_{i=1}^{d} \alpha_i \\ (1-p)^d &\geq \sum_{i=1}^{d} p(1-p)^{d-i} \\ (1-p)^d &\geq 1-(1-p)^d \\ p &\leq 1-2^{-1/d}. \end{aligned} \tag{5}$$

Por exemplo, para $d = 10$ tem-se $p \leq 0,067$. Isto mostra que o atacante pode forjar m caminhos diferentes com a mesma probabilidade que o caminho original, como mostra a Figura 1.18.

Se o valor de m for grande, ou seja, se o atacante for capaz de forjar muitos caminhos, a vítima terá problemas para identificar o verdadeiro caminho. No entanto, o valor de m é limitado por $(d-1)$ [Park e Lee, 2001]. Também é mostrado que um aumento no valor de p diminui o valor de m. Desta maneira, Park e Lee demonstram que mecanismos baseados em PPM podem ser eficientes para ataques de apenas um atacante, mas são vulneráveis a ataques distribuídos com múltiplos atacantes. Isto ocorre, pois a maneira de maximizar o número de caminhos forjados m é forçar a diminuição do valor

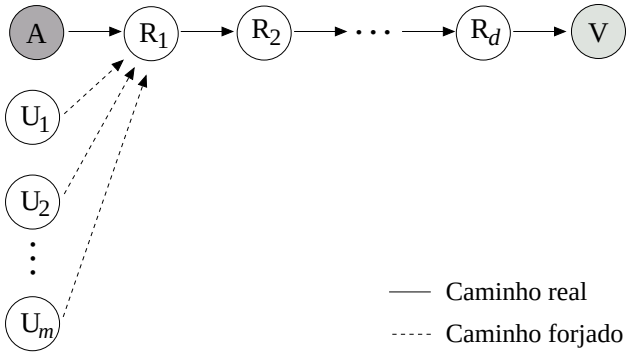

Figura 1.18. Caminhos forjados pelo atacante.

de *p*. A única possibilidade seria diminuir o número de pacotes *N* enviados pelo atacante. Isto porque *N* deve ser suficiente para que a vítima possa receber pelo menos um pacote do roteador R_1. Entretanto, o valor de *N* não é pequeno em ataques de negação de serviço por inundação. Assim, no caso de apenas um atacante, é mostrado que com *N* entre 10^3 e 10^5 pacotes e $d = 25$, o valor de *m* fica limitado entre dois e cinco. Porém, em um ataque com múltiplos atacantes, onde cada atacante possa reduzir significativamente o número de pacotes enviados, o valor de *m* tenderá à $(d-1)$, que no caso de $d = 25$ é uma quantidade significativa de caminhos possíveis para cada atacante.

Com objetivo de diminuir a complexidade do algoritmo de reconstrução de rota e de aumentar a confiabilidade a fim de permitir o rastreamento de um ataque distribuído com um grande número de atacantes, um novo esquema de rastreamento foi proposto [Goodrich, 2002]. Nesta nova proposta, foi introduzido o conceito de mensagem (M_R) que será enviada pelo roteador *R* à vítima. O conteúdo da mensagem depende do seu tamanho. Mas a princípio, a mensagem mais simples contém o endereço IP do roteador em questão, sendo possível acrescentar outras informações, como por exemplo, o endereço do próximo roteador e até mesmo informações para autenticação. O processo de marcação de pacote é basicamente o mesmo, com a diferença que não existe mais o campo *distância*. Isto significa que quando o pacote não é marcado, não há sobrecarga para o roteador. A codificação da mensagem no cabeçalho IP é feita utilizando 25 bits. Primeiramente, a mensagem é preenchida, se necessário, para que seu tamanho seja múltiplo de *l*. Em seguida é gerada uma soma de verificação (*checksum*) de *c* bits da mensagem ($C(M_R)$). Depois a mensagem é divida em *l* partes iguais, denominadas de palavras, w_0, $w_1, \ldots, w_{l-1}$. Os 25 bits do cabeçalho são divididos, como mostra a Figura 1.19.

índice	$C(M_R)$	palavra (w_i)	
$\log_2 l$	c	$25 - c - \log_2 l$	bits

Figura 1.19. Marcação do cabeçalho segundo Goodrich.

Ao marcar um pacote, o roteador deve escrever uma das palavras da mensagem no campo *palavra*, o valor de $C(M_R)$, no campo *checksum* e o índice da palavra no campo

índice. A reconstrução é bastante similar a de Savage *et al*. A diferença está na remontagem da mensagem. O *checksum* além de ser o responsável por garantir se a mensagem foi corretamente montada, serve também para identificar todas as palavras da mesma mensagem. Deste modo, basta que a vítima separe todos os pacotes que possuam o mesmo *checksum*, eliminar as repetições e testar todas as combinações possíveis com as palavras de mesmo índice e mesmo *checksum*. Goodrich demonstra que seu mecanismo apresenta uma complexidade bastante inferior às propostas de Savage *et al.* e Song e Perrig. Outro fator importante é o fato de conseguir rastrear ataques com até 1000 atacantes. Por último, o esquema de mensagem é flexível, permitindo acrescentar informações para autenticação dos roteadores.

Após diversos trabalhos na área de marcação de pacotes, percebeu-se que o grande problema dos mecanismos baseados na PPM é o fato dos atacantes poderem forjar uma marcação. Assim, acabar com a possibilidade do atacante forjar uma marcação, resolveria o principal inconveniente deste tipo de abordagem. Por isso, foi proposto um novo mecanismo de marcação de pacotes determinístico que utiliza 17 bits (16 do campo *Packet ID* e um do campo reservado *flag*) do cabeçalho IP [Belenky e Ansari, 2003b]. O mecanismo pressupõe que a marcação seja feita apenas pelos roteadores de borda. Desta maneira, todo pacote que entrar por uma interface de "borda" será marcado com o endereço IP da respectiva interface. Como são usados apenas 17 bits, o endereço da interface é divido em duas partes. O roteador deve escrever no pacote uma das duas partes com igual probabilidade. Em seguida deve-se acrescentar zero ou um ao campo *flag*, de acordo com a parte do endereço escrita no campo *Packet ID*. Assim, após a recepção de alguns pacotes, a vítima obterá as duas partes do endereço da interface de entrada do roteador mais próximo do atacante. Com apenas 7 pacotes a vítima tem 99% de chance de obter o endereço completo. Um dos problemas desta abordagem é a fragmentação de pacotes, pois todos os fragmentos de um determinado pacote devem conter o mesmo valor para o campo *Packet ID* para serem corretamente remontados. Isto pode não acontecer, dado que dois valores distintos podem ser escritos no campo *Packet ID*. Este problema foi resolvido identificando os fragmentos e escrevendo a mesma parte do endereço da interface de entrada no campo *Packet ID* [Belenky e Ansari, 2003a]. Entretanto, esta abordagem possui dois problemas graves que podem impedir que a vítima descubra o endereço da interface de entrada do roteador de borda. O primeiro problema aparece quando o atacante envia apenas um pacote com cada endereço IP. O segundo caso seria um ataque distribuído onde m atacantes enviam pacotes com os mesmos endereços IP. Assim, para cada endereço IP diferente a vítima receberia m pacotes para cada parte do endereço.

Ainda na tentativa de impedir o atacante de forjar as marcações, foi desenvolvido um outro mecanismo de rastreamento baseado na marcação de pacotes, denominado *Dynamic Probabilistic Packet Marking* (DPPM) [Liu et al., 2003]. Seu objetivo é minimizar a capacidade do atacante de forjar caminhos, que, como foi mostrado por Park e Lee [Park e Lee, 2001], está diretamente relacionada com o valor de p e com o número N de pacotes enviados pelo atacante. Como este último fator é determinado pelo atacante, a única opção é mudar o valor de p. Desta forma, a proposta consiste em atribuir diferentes probabilidades de marcação de acordo com a distância que o pacote já percorreu. Assim, para um caminho com d roteadores, o pacote será marcado pelo roteador R_i com a probabilidade $p_i = 1/i$. A probabilidade de um pacote chegar marcado pelo roteador R_i na

vítima é dado pela seguinte expressão

$$\alpha_i = p_i \prod_{j=i+1}^{d} 1 - p_j. \tag{6}$$

A Equação 1 é um caso particular da Equação 6, onde $p_i = p$, para $1 \leq i \leq d$. Expandindo a Equação 6 e substituindo $p_i = 1/i$, tem-se a seguinte expressão:

$$\begin{aligned}
\alpha_i &= p_i \prod_{j=i+1}^{d} 1 - p_j \\
&= p_i(1-p_{i+1})(1-p_{i+2})\ldots(1-p_d) \\
&= \left(\frac{1}{i}\right)\left(1-\frac{1}{i+1}\right)\left(1-\frac{1}{i+2}\right)\ldots\left(1-\frac{1}{d}\right) \\
&= \frac{1}{d}.
\end{aligned} \tag{7}$$

Isto significa que a probabilidade da vítima receber um pacote marcado pelo roteador R_i é a mesma para todos os roteadores, não importando a distância para a vítima. Este resultado é bastante interessante, pois a probabilidade do pacote chegar marcado pelo atacante α_0, ou seja, a probabilidade do atacante forjar uma marcação é igual à zero. Intuitivamente, pode-se ver este resultado a partir da seguinte expressão:

$$\alpha_0 = 1 - \sum_{i=1}^{d} \alpha_i = 0 \tag{8}$$

Além disso, o valor mínimo de pacotes necessários para a reconstrução da rota é igual à d, pois $N \cdot \min(\alpha_i) \geq 1$. Na PPM, $\min(\alpha_i) = \alpha_1$, mas no DPPM todos os α_i são iguais à $1/d$. Logo,

$$\begin{aligned}
N \cdot \frac{1}{d} &\geq 1 \\
N &\geq d.
\end{aligned} \tag{9}$$

O grande problema desta abordagem é saber quantos nós o pacote já percorreu. Uma das possíveis soluções é a utilização do campo TTL (*Time To Live*) do cabeçalho IP para a contagem dos saltos de distância do atacante. No entanto, o atacante pode facilmente forjar o TTL para tentar burlar o mecanismo de rastreamento de pacote. Por isso, os autores sugerem a utilização de um valor máximo de TTL inicial (TTL_{max}) para um pacote, de maneira que se um roteador recebe um pacote com $TTL > TTL_{max}$, ele deve sobrescrever o valor do TTL com TTL_{max}. O valor sugerido para o TTL máximo é de 32 ou 64, considerando-se que a maioria das rotas da Internet possuem menos que 25 roteadores [Liu et al., 2003]. Neste caso, se o atacante escolhe um valor menor para o campo TTL, como $TTL = TTL_{max} - z$, a probabilidade p_i de um roteador marcar um determinado pacote será

$$p_i = \frac{1}{z+i}. \tag{10}$$

Substituindo este novo valor na Equação 6, obtém-se a probabilidade do pacote chegar a vítima marcado pelo roteador R_i,

$$\alpha_i = \frac{1}{z+d}. \tag{11}$$

Assim, a probabilidade de chegar à vítima um pacote marcado pelo atacante pode ser escrito da seguinte maneira:

$$\begin{aligned} \alpha_0 &= 1 - \sum_{i=1}^{d} \alpha_i \\ &= 1 - \frac{d}{z+d} \\ &= \frac{z}{z+d}. \end{aligned} \tag{12}$$

Esta não aparenta ser uma boa solução. Por exemplo, no caso de $TTL_{max} = 64$ e o atacante ter marcado um TTL = 28 ($z = 36$), α_0 será igual a $0,36$. De acordo com os próprios autores, este valor de TTL ainda é suficiente para alcançar a grande maioria dos sítios da Internet. O atacante poderia ainda diminuir o valor do TTL, de acordo com a distância da vítima, piorando ainda mais a eficiência do mecanismo de rastreamento.

Além da preocupação com a capacidade do atacante de forjar arestas, existe ainda um grande interesse na otimização da codificação das arestas, para tornar o rastreamento baseando na marcação de pacotes um sistema mais eficiente. Um mecanismo denominado SNITCH [Aljifri et al., 2003] é uma extensão a PPM na qual é modificada a codificação da identificação das arestas. Neste trabalho é proposta a utilização de uma técnica de compressão do cabeçalho IP, definida na RFC 2507 [M.Degermark et al., 1999] e na RFC 1144 [Jacobson, 1990]. O objetivo desta técnica de compressão do cabeçalho IP é diminuir o tamanho do pacote para aumentar a vazão de enlaces de baixa velocidade. Esta técnica pressupõe a existência de campos no cabeçalho IP que não mudam em uma seqüência de pacotes do mesmo fluxo. Estes campos são denominados de contexto do pacote. Assim, apenas o primeiro pacote enviado conterá todos os campos do cabeçalho que será associado a um contexto. Os pacotes seguintes, que possuam o mesmo contexto, terão estes campos removidos do cabeçalho e armazenarão o identificador do contexto ao qual estão associados (*Context ID* – CID). O espaço economizado com esta técnica é igual a 144 bits. Na proposta de Aljifri *et al.* [Aljifri et al., 2003], estes 144 bits são utilizados para a marcação dos pacotes. Desta maneira, ao decidir marcar um pacote, o roteador associa o contexto deste pacote a um CID. Em seguida, o roteador compara o contexto do primeiro pacote com o pacote seguinte. Caso os contextos sejam idênticos, o roteador comprime este cabeçalho e acrescenta as informações relativas ao mecanismo de rastreamento. A vítima, ao receber este pacote, retira as informações de rastreamento antes de descomprimir o cabeçalho. Após coletar o número suficiente de pacotes, as duplicatas são eliminadas e a rota pode ser construída a partir das arestas e das distâncias. Apesar de conseguir aumentar a quantidade de dados escritos no cabeçalho, evitando a fragmentação das informações de rastreamento, a proposta apresenta o mesmo problema da alta probabilidade do atacante forjar caminhos.

Uma alternativa para a codificação das informações de rastreamento é apresentada por Dean *et al.* [Dean et al., 2002]. A codificação é baseada em equações algébricas. As-

sim, cada roteador indica sua presença na rota de ataque, acrescentando o resultado de um polinômio no qual a variável é o seu endereço IP. A vítima deve ter conhecimento deste polinômio. Após a recepção de um número suficiente de pacotes, a reconstrução da rota de ataque é feita a partir da resolução de um sistema de equações. No entanto, a proposta também é baseada na marcação de pacotes e, por conseqüência, possui as mesmas desvantagens em relação à capacidade do atacante forjar marcações. Além disso, diferente do sistema proposto por Savage *et al.*, este sistema não apresenta nenhum código de detecção de erro para reduzir a probabilidade de se construir um sistema de equações utilizando equações provenientes de rotas diferentes. Conseqüentemente, ainda mais falsos positivos são esperados para ataques distribuídos de pequeno porte. Um outro tipo de equações algébricas foi sugerido na tentativa de diminuir a quantidade de bits usados para a codificação [Bai et al., 2004].

Existe ainda outro método baseado em auditoria, que não se baseia na marcação de pacotes [Bellovin et al., 2003]. Ao rotear um pacote, cada roteador probabilisticamente envia para a vítima um pacote ICMP com informações sobre si mesmo e sobre seus roteadores adjacentes. Para um fluxo suficientemente longo, a vítima usa estes dados recebidos para reconstruir a rota de ataque. Entretanto, como as informações de auditoria são enviadas em pacotes separados, a autenticação das mensagens é necessária de forma a evitar mensagens forjadas pelo atacante. Logo, a adoção de uma infra-estrutura de chave pública se torna inevitável. Uma extensão desta idéia inclui novos conceitos como "utilidade" e "valor" das mensagens de rastreamento [Mankin et al., 2001], a fim melhorar o sistema.

Armazenamento na Infra-estrutura de Rede

Sob a perspectiva de armazenamento na própria infra-estrutura de rede, a maneira mais simples de recolher rastros de auditoria é cada roteador registrar todos os pacotes que o atravessam [Stone, 2000]. Porém, recursos excessivos são requisitados tanto para armazenagem quanto para a mineração dos dados. Além disso, a invasão de um roteador acarretaria ainda em problemas de privacidade, uma vez que ele contém informações sobre todos os pacotes roteados.

Uma alternativa para reduzir a armazenagem de um grande volume de informações é utilizar um Filtro de Bloom [Bloom, 1970]. Ultimamente, estes filtros têm sido amplamente usados em redes de computadores [Broder e Mitzenmacher, 2003].

O Filtro de Bloom [Bloom, 1970] é uma estrutura de dados usada para representar de forma compacta um conjunto $S = \{s_1, s_2, \ldots, s_n\}$ de n elementos. Ele é constituído por um vetor de m bits e por k funções *hash* independentes $h_1, h_2, \ldots, h_k$ cujas saídas variam uniformemente no espaço discreto $\{0, 1, \ldots, m-1\}$. O vetor de bits é obtido da seguinte forma. Inicialmente, todos os seus bits encontram-se zerados. Para cada elemento $s_i \in S$, os bits do vetor correspondentes às posições $h_1(s_i), h_2(s_i), \ldots, h_k(s_i)$ são preenchidos com 1. O mesmo bit pode ser preenchido diversas vezes sem restrições. A Figura 1.20 ilustra a inserção de um elemento no filtro de maneira sucinta. Uma vez que o Filtro de Bloom é uma forma compacta de representar um conjunto de elementos, testes de pertinência podem ser realizados visando determinar se um elemento x pertence ou não ao conjunto S. Para isso, verifica-se se os bits do vetor correspondentes às posições

$h_1(x), h_2(x), \ldots, h_k(x)$ estão preenchidos com 1. Se pelo menos um bit estiver zerado, então com certeza $x \notin S$. Por outro lado, se todos os bits estão preenchidos, então assume-se que $x \in S$. Na verdade, um elemento externo $x \notin S$ pode ser reconhecido como um autêntico elemento do conjunto, criando um falso positivo. Tal anomalia ocorre quando todos os bits $h_1(x), h_2(x), \ldots, h_k(x)$ são preenchidos por um subconjunto dos elementos de S.

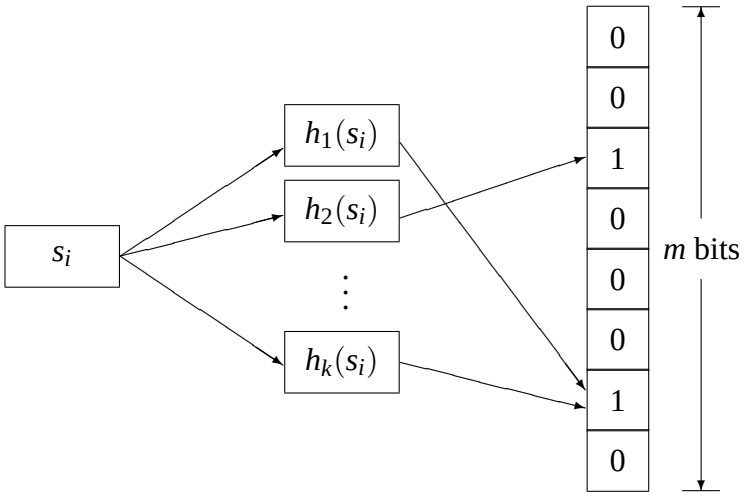

Figura 1.20. Inserção de um elemento em um Filtro de Bloom.

A probabilidade de se encontrar um falso positivo para um elemento $x \notin S$ é calculada de maneira trivial. Dado que as funções *hash* usadas são uniformes e independentes, a probabilidade p de um determinado bit permanecer em zero mesmo depois de inseridos os n elementos é

$$p = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}. \tag{13}$$

Como o mesmo cálculo se aplica para todos os bits do vetor, na média, uma fração $e^{-kn/m}$ dos bits permanece zerada após as inserções [Mitzenmacher, 2002]. Desta forma, a fração média de bits preenchidos com 1 depois de n inserções é $\left(1 - e^{-kn/m}\right)$. A probabilidade de falso positivo f é então a probabilidade de se encontrar um bit em 1 para cada uma das k posições indicadas, ou seja

$$f = \left(1 - e^{-kn/m}\right)^k. \tag{14}$$

A utilização do Filtro de Bloom no sistema de rastreamento permitiu a criação de um mecanismo que possui a vantagem de rastrear um único pacote IP que tenha passado na rede sem a necessidade de se armazenar todo o tráfego roteado [Snoeren et al., 2002]. Para isso, são usados Filtros de Bloom em dispositivos acoplados aos roteadores que armazenam os pacotes roteados de forma compacta. Periodicamente, os filtros saturados são armazenados para futuras requisições e trocados por novos filtros vazios. Para mais tarde determinar se um pacote passou pelo roteador, o seu filtro simplesmente é verificado. Um processo repetitivo pode ser feito por cada roteador para reconstruir o caminho do

pacote até a sua verdadeira origem. Porém, mesmo com o uso de Filtros de Bloom, tal sistema exige uma alta capacidade de armazenamento.

Um outro mecanismo baseado no armazenamento na infra-estrutura de rede consiste em cada roteador amostrar uma pequena porcentagem do tráfego e armazenar os sumários dos pacotes amostrados [Li et al., 2004]. O compromisso da amostragem é a dificuldade inserida no processo de rastreamento. Para minimizar o número de pacotes necessários para rastrear a origem de um ataque com acurácia, é necessário aumentar o fator de correlação entre roteadores vizinhos. Este fator representa a porcentagem dos pacotes amostrados por um roteador que também foram amostrados pelo roteador seguinte. Além disso, os autores introduzem um arcabouço que utiliza a teoria da informação para responder algumas perguntas sobre a escolha de parâmetros do sistema e sobre o compromisso entre os recursos necessários e a acurácia do rastreamento.

O método de amostragem independente não apresenta um bom funcionamento já que o fator de correlação dos pacotes amostrados por roteadores vizinhos é somente p, o mesmo valor da taxa de amostragem. Para aumentar este fator, um outro método é abordado pelos autores. A idéia principal deste método é que, além de amostrar o pacote, o roteador também marca o pacote amostrado de modo que o próximo roteador possa coordenar a sua marcação com a do roteador anterior. O campo necessário para a marcação é de apenas 1 bit e pode se localizar, por exemplo, no campo *reserved flag*, não utilizado do cabeçalho IP.

O processo de rastreamento se inicia com a vítima verificando seus roteadores vizinhos. Para cada roteador S, a vítima pede a verificação da passagem dos pacotes do conjunto L_v (conjunto de pacotes de ataque selecionados pela vítima) nos seus Filtros de Bloom correspondentes. Se pelo menos um pacote é reconhecido, o roteador S monta um conjunto de pacotes L_S a partir dos pacotes em L_v que foram encontrados em S. Agora, cada roteador R vizinho de S será verificado com L_S. Sendo R "condenado", todo o conjunto L_v é novamente repassado para o roteador para a formação de um conjunto L_R. Este processo é repetido recursivamente para cada roteador. Através da elaboração de um arcabouço de teoria da informação, os autores chegam a conclusão de que a utilização de $k = 12$ (k é o número de funções *hash* usadas no Filtro de Bloom) é um valor ótimo para o desempenho do método de rastreamento. As melhorias propostas por Li *et al.* [Li et al., 2004] diminuem drasticamente o espaço necessário para o armazenamento embora a capacidade de se rastrear um único pacote seja comprometida.

Baseado ainda na técnica de utilização de funções *hash* para o armazenamento das informações de rastreamento nos roteadores da rede [Snoeren et al., 2001], foi proposto um novo mecanismo que armazena as informações referentes aos fluxos a fim de reduzir a quantidade de informações armazenadas [Lee et al., 2004]. Uma outra melhoria foi proposta para melhorar a acurácia do grafo reconstruído [Hilgenstieler e Duarte Jr., 2004].

Armazenamento Híbrido

Algumas propostas visam conciliar as vantagens do armazenamento nas estações finais com as do armazenamento na infra-estrutura de redes, a fim de minimizar os problemas intrínsecos a cada uma das abordagens. Com este objetivo, é proposto o armaze-

namento híbrido, que consiste em armazenar parte das informações na vítima e parte nos roteadores da rede.

Dentro deste contexto, foi proposto um novo mecanismo que combina a marcação de pacotes com o armazenamento de informações nos roteadores [Choi e Dai, 2004]. O esquema consiste em acrescentar ao cabeçalho o enlace pelo qual o pacote entrou. Para isto, é reservada uma parte do cabeçalho IP de 32 bits para armazenar as informações. O primeiro bit é definido como o campo *save flag* que indica se as informações de rastreamento referentes àquele pacote foram armazenadas em algum roteador. Os 31 bits restantes são utilizados para guardar os enlaces. Este campo é sempre iniciado com o primeiro bit em 1 e os outros em zero, pois o último bit em 1 indica o fim do último enlace acrescentado, como apresentado na Figura 1.21.

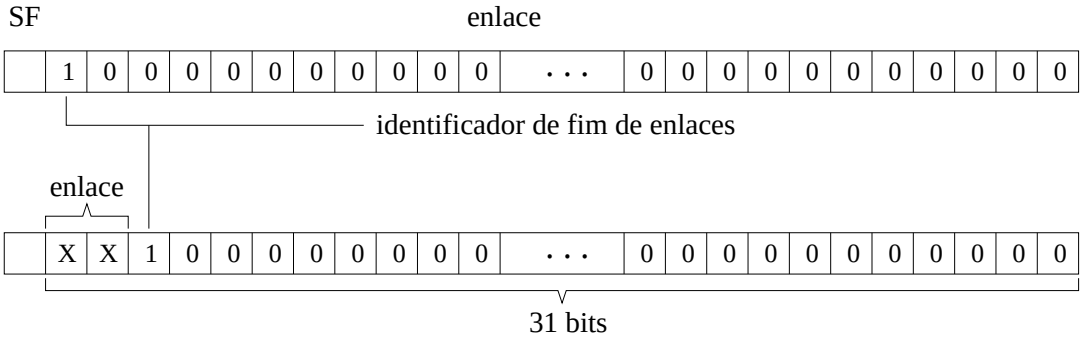

Figura 1.21. Codificação do esquema de marcação utilizando armazenamento híbrido.

Assim, ao perceber que não há mais espaço no cabeçalho para escrever o enlace, o roteador dever criar um identificador para o pacote a partir da aplicação de uma função *hash* em uma parte estática do pacote, ou seja, uma parte que não irá mudar até chegar a vítima. Em seguida, o nó deve armazenar todo o campo *enlace* em sua memória e reiniciá-lo. Por último, é preciso fazer a marcação e atualização do campo *save flag* com o valor 1. Para economizar espaço e, por conseqüência, minimizar o armazenamento de informações nos roteadores, os enlaces são representados por palavras geradas a partir do código de Huffman. O princípio de funcionamento deste código é representar as informações mais freqüentes com uma quantidade menor de bits. Desta maneira, cada roteador armazena uma tabela de enlaces, que contém todos os enlaces com seus vizinhos. Cada enlace será associado a uma freqüência que será utilizada para obter a sua codificação. A freqüência de um enlace é definida pela quantidade de pacotes que o atravessam. O objetivo é evitar ao máximo o armazenamento de informações nos roteadores. A eficiência deste mecanismo depende, basicamente, do número de vizinhos de cada roteador e do número de roteadores no caminho. Choi e Dai consideram que o número médio de vizinhos na Internet é 3,15. Pare este valor, a média de bits utilizados para representar os enlaces é 2. Isto significa que até 15 nós (15×2) não é preciso o armazenamento nos roteadores. Uma rota com mais de 15 nós será armazena pelo menos uma vez em um roteador intermediário. Também é mostrado que a quantidade de informação armazenada é inferior à quantidade das propostas baseadas em *hash*.

A reconstrução da rota de ataque começa a partir do roteador que está diretamente

conectado a vítima. O primeiro passo é consultar a tabela de enlaces deste roteador para identificar a que enlace a primeira palavra se refere. Em seguida, deve-se fazer um deslocamento para esquerda de *n* bits, onde *n* representa o tamanho da palavra recém decodificada. Este procedimento será realizado até que o campo enlace retorne a sua condição inicial (primeiro bit em 1 e o restante em zero). Neste momento, é necessário checar o campo *save flag*. O valor deste campo em 0 significa que o procedimento de reconstrução chegou ao fim. O valor deste campo em 1 significa que existe informação sobre este pacote armazenada no roteador em questão. Por isso, o nó deve recuperar o campo enlace armazenado e continuar o procedimento de reconstrução.

1.5. Um Novo Sistema de Rastreamento de Pacotes IP

Esta seção apresenta brevemente o sistema de rastreamento de pacotes IP proposto [Laufer et al., 2005c]. Esse sistema objetiva rastrear a origem de cada pacote individualmente. Ele é baseado na inserção de informações nos pacotes para evitar o armazenamento na infra-estrutura da rede. Resumidamente, cada roteador insere no pacote uma "assinatura" que indica a sua presença na rota. Ao receber um pacote de ataque, a vítima usa as marcações feitas pelos roteadores para reconstruir a rota reversa. Para diminuir o espaço necessário no cabeçalho e o custo de processamento, um Filtro de Bloom é embutido em cada pacote para armazenar a rota de ataque. Para isso, um campo fixo é reservado para o filtro no cabeçalho do pacote. A técnica de Filtro de Bloom é usada para que a quantidade de informações inserida seja reduzida e permaneça constante. Um tamanho constante para as marcações é importante para evitar tanto a fragmentação dos pacotes quanto o processamento resultante da adição de dados. Além disso, uma generalização do Filtro de Bloom foi proposta para evitar que o atacante possa forjar as "assinaturas" e prejudicar o rastreamento.

O procedimento de marcação para o sistema de rastreamento é bem simples e ocorre pouco antes de reencaminhar um pacote. Neste momento, o roteador insere no filtro daquele pacote o endereço IP da sua interface de saída. Dessa forma, ao receber um pacote de ataque, a vítima dispõe de um filtro cujos elementos são os endereços de todos os roteadores da rota de ataque. Uma vantagem importante desse procedimento de marcação é o seu baixo processamento adicional. Na verdade, o cálculo dos valores *hash* não precisa ser realizado para cada pacote roteado. Os valores *hash* do endereço IP de cada interface do roteador podem ser calculados inicialmente e armazenados em um registrador existente para cada interface. Esse registrador pode ser interpretado como um Filtro de Bloom com somente um elemento inserido: o endereço IP da interface. Quando um pacote vai ser encaminhado, o seu filtro é atualizado com o resultado de uma operação OU bit-a-bit do próprio filtro do pacote com o registrador da interface de saída do roteador. Dessa forma, o endereço IP da interface de saída do roteador é adicionado ao filtro do pacote de maneira eficiente.

Para reconstruir a rota de ataque, o seguinte algoritmo é utilizado. Inicialmente, a vítima verifica quais dos seus roteadores vizinhos estão presentes no filtro do pacote recebido. Aquele que for reconhecido como elemento do filtro é identificado como o roteador pelo qual o pacote chegou e é, portanto, integrado à rota de ataque. Em seguida, o roteador vizinho recebe o filtro da vítima de forma a continuar o procedimento de reconstrução. Ele então verifica qual dos seus roteadores vizinhos também é reconhecido

como elemento do filtro, identificando assim o próximo componente da rota de ataque. O processo é então realizado sucessivamente por cada roteador visando reconstruir o caminho do pacote até a sua verdadeira origem. Quando nenhum roteador é reconhecido, o procedimento termina e o último roteador identificado é considerado a fonte do ataque. A Figura 1.22 ilustra a reconstrução de rota iniciada pela vítima V em direção ao atacante A. Inicialmente, o atacante envia um pacote para a vítima que passa por (R_5, R_4, R_2, R_1). Ao receber o pacote de ataque, a vítima inicia o procedimento de reconstrução, testando a presença de R_1 no filtro do pacote recebido (1). Como R_1 é reconhecido, ele recebe o filtro de V e continua o procedimento. Assim, R_1 verifica a presença dos seus vizinhos R_2 e R_3 no filtro (2). Como somente R_2 é reconhecido, o filtro é então repassado somente para R_2, que faz o mesmo procedimento com seu vizinho R_4 (3). O roteador R_4 verifica qual dos seus dois vizinhos R_3 e R_5 é reconhecido pelo filtro (4); somente R_5 é reconhecido. Finalmente, R_5 testa a presença de R_7 no filtro (5). Uma resposta negativa é retornada e o procedimento de reconstrução termina.

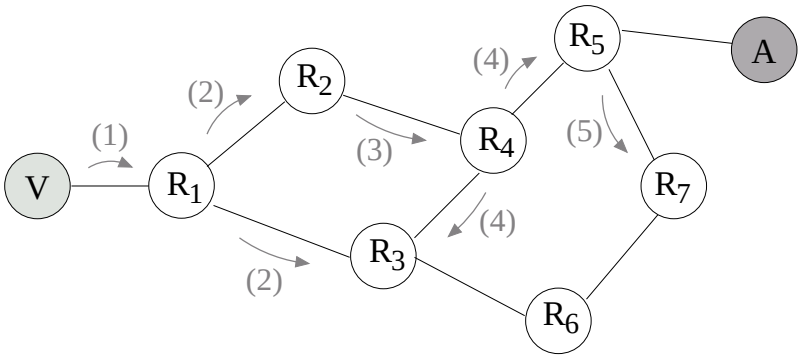

Figura 1.22. Exemplo do procedimento de reconstrução de rota.

Algumas vantagens surgem como resultado da adoção dessa abordagem. Em primeiro lugar, a rota completa de cada pacote pode ser determinada individualmente. Tal comportamento é idealizado por todo sistema de rastreamento de pacotes, uma vez que possibilita a identificação de qualquer fonte em um ataque distribuído. Além disso, nenhuma informação é armazenada na infra-estrutura de rede. Todos os dados relativos ao rastreamento estão localizados na própria vítima, que opta por guardá-los ou não de acordo com a política de segurança local. Outra vantagem é que o sistema proposto não só evita o processamento resultante da fragmentação e da adição de dados ao pacote como também introduz muito pouco processamento adicional ao roteamento. Na verdade, somente uma operação OU bit-a-bit é adicionada ao processo de roteamento. Além disso, é possível realizar o rastreamento após o término do ataque e sem ajuda de operadores de rede. No caso, todo o procedimento de reconstrução pode ser automatizado e tornar-se independente de intervenções manuais.

Por outro lado, esta abordagem também possui desvantagens presentes em outros sistemas de rastreamento [Bellovin et al., 2003], [Dean et al., 2002], [Savage et al., 2001], [Snoeren et al., 2002], [Song e Perrig, 2001]. Primeiramente, um processamento adicional é introduzido ao roteamento de pacotes. Embora o sistema proposto introduza menos processamento que outros sistemas, roteadores com poucos recursos podem ser afetados. Além disso, como em qualquer outro sistema de rastreamento, a cooperação dos roteado-

res é fundamental para a correta marcação dos pacotes. Se alguns roteadores não marcam os pacotes, é provável que existam lacunas na rota reconstruída e que a origem do ataque não seja encontrada. Uma outra desvantagem é que o próprio atacante não é encontrado pelo rastreamento; na verdade, somente o roteador mais próximo do atacante é revelado. Após a identificação deste roteador, maiores esforços são necessários para identificar o atacante. Por fim, a adoção de um Filtro de Bloom para representar a rota de ataque introduz uma certa probabilidade de falso positivo. Durante o algoritmo de reconstrução, um falso positivo implica a incorreta integração de um roteador à rota de ataque. Porém, se esta probabilidade é pequena, a ocorrência de falsos positivos não tem impacto significativo na reconstrução. Algumas rotas para um mesmo pacote existiriam em paralelo, mas ainda assim o escopo de possíveis atacantes seria restringido. No entanto, como o atacante tem controle sobre o conteúdo inicial do pacote, ele pode preencher com 1 todos os bits do cabeçalho do pacote que são reservados ao filtro. Ao saturar o filtro, o atacante faz com que a vítima receba um filtro cujos bits estão todos preenchidos com 1. Conseqüentemente, todo roteador é integrado à rota de ataque durante a reconstrução, tornando impraticável a descoberta da rota verdadeira.

Para minimizar a possibilidade do atacante burlar o sistema e torná-lo menos dependente da condição inicial do filtro, uma generalização do Filtro de Bloom também é proposta. O chamado Filtro de Bloom Generalizado (FBG) estaria integrado a cada pacote, de forma a armazenar os roteadores atravessados. A idéia básica do FBG é utilizar tanto funções *hash* que preenchem bits como funções que zeram bits. Desta forma, é possível mostrar que a probabilidade de falso positivo é limitada e depende pouco da condição inicial do filtro. Por outro lado, falsos negativos que eram inexistentes no Filtro de Bloom convencional são introduzidos com esta generalização. Na Subseção 1.5.1, a idéia do FBG é formalizada e uma análise das probabilidades de falso negativo e falso positivo é desenvolvida a fim de mostrar a eficácia dessa nova abordagem.

1.5.1. O Filtro de Bloom Generalizado

Assim como o filtro convencional, o Filtro de Bloom Generalizado é uma estrutura de dados usada para representar de forma compacta um conjunto $S = \{s_1, s_2, \ldots, s_n\}$ de n elementos. Ele é constituído por um vetor de m bits e por $k_0 + k_1$ funções *hash* independentes $g_1, g_2, \ldots, g_{k_0}, h_1, h_2, \ldots, h_{k_1}$ cujas saídas variam uniformemente no espaço discreto $\{0, 1, \ldots, m-1\}$. O vetor de bits é calculado de maneira semelhante ao do caso convencional. Entretanto, não há mais a restrição de que seus bits são inicialmente zerados. No FBG, os bits do vetor podem assumir qualquer valor inicial. Para cada elemento $s_i \in S$, os bits do vetor correspondentes às posições $g_1(s_i), g_2(s_i), \ldots, g_{k_0}(s_i)$ são zerados e os bits correspondentes às posições $h_1(s_i), h_2(s_i), \ldots, h_{k_1}(s_i)$ são preenchidos com 1. No caso de uma colisão entre uma função g_i com uma função h_j em um mesmo elemento, arbitra-se que o bit é zerado $\forall i, j$. O mesmo bit pode ser zerado ou preenchido diversas vezes sem restrições. A Figura 1.23 ilustra a inserção de um elemento em um filtro generalizado. Posteriormente, testes de pertinência podem ser realizados visando determinar a afiliação de um elemento ao conjunto. Para verificar se um elemento x pertence a S, é preciso checar se os bits $g_1(x), g_2(x), \ldots, g_{k_0}(x)$ estão zerados e se os bits $h_1(x), h_2(x), \ldots, h_{k_1}(x)$ estão preenchidos com 1. Se pelo menos um bit está invertido, então assume-se que $x \notin S$. Diferentemente do filtro convencional, agora há uma possibilidade de um elemento $x \in S$ não

ser reconhecido pelo filtro, criando um falso negativo. Tal anomalia ocorre quando pelo menos um dos bits $g_1(x), g_2(x), \ldots, g_{k_0}(x)$ é preenchido ou quando pelo menos um dos bits $h_1(x), h_2(x), \ldots, h_{k_1}(x)$ é zerado por algum outro elemento inserido posteriormente. Por outro lado, se nenhum bit está invertido, então assume-se que $x \in S$. Na verdade, um elemento externo $x \notin S$ pode ser reconhecido como um autêntico elemento do conjunto, criando um falso positivo. Um falso positivo ocorre quando os bits $g_1(x), g_2(x), \ldots, g_{k_0}(x)$ estão zerados e os bits $h_1(x), h_2(x), \ldots, h_{k_1}(x)$ estão preenchidos com 1 em virtude de um subconjunto dos elementos de S ou da própria condição inicial do vetor.

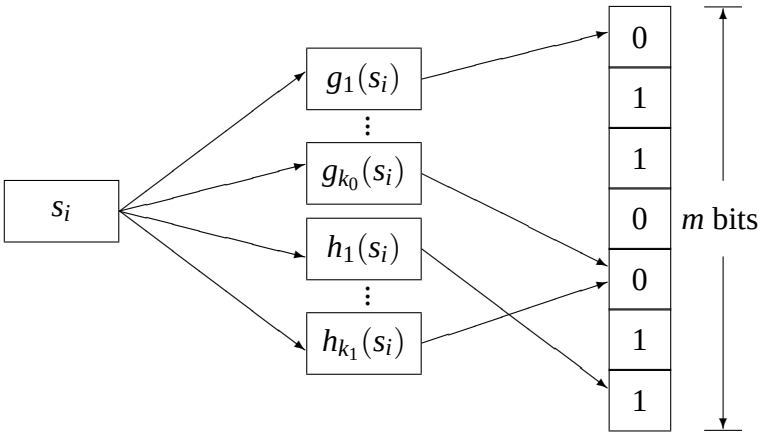

Figura 1.23. Inserção de um elemento em um Filtro de Bloom Generalizado.

Falsos Positivos

A probabilidade de falso positivo de um FBG é calculada de maneira similar à do caso convencional. Entretanto, é necessário primeiramente calcular a probabilidade de um bit ser preenchido ou zerado durante a inserção de um elemento. Dado que no caso de uma colisão as funções g_i sempre têm prioridade sobre as funções h_j, a probabilidade q_0 de um determinado bit ser zerado durante a inserção de um elemento é expressa pela probabilidade de pelo menos uma das k_0 funções zerar o bit. Assim, q_0 é expresso por

$$q_0 = \left[1 - \left(1 - \frac{1}{m}\right)^{k_0}\right] \approx \left(1 - e^{-k_0/m}\right). \tag{15}$$

Por outro lado, a probabilidade q_1 de um determinado bit ser preenchido durante a inserção de um elemento é a probabilidade de pelo menos uma das k_1 funções preencher o bit e nenhuma das k_0 funções o zerar. Dessa forma, q_1 é definido como

$$q_1 = \left[1 - \left(1 - \frac{1}{m}\right)^{k_1}\right]\left(1 - \frac{1}{m}\right)^{k_0} \approx \left(1 - e^{-k_1/m}\right)e^{-k_0/m}. \tag{16}$$

Por fim, a probabilidade de um bit específico permanecer intocado, isto é, não ser nem preenchido e nem zerado por um elemento, é simplesmente

$$(1-q_0-q_1) = \left(1-\frac{1}{m}\right)^{k_0+k_1} \approx e^{-(k_0+k_1)/m}. \quad (17)$$

Como o mesmo cálculo se aplica para todos os bits do vetor, na média, uma fração q_0 de bits é zerada, uma fração q_1 de bits é preenchida e uma fração $(1-q_0-q_1)$ de bits permanece intocada a cada inserção de elemento. Seguindo o mesmo raciocínio, tem-se na média $b_0 = m.q_0$ bits zerados, $b_1 = m.q_1$ bits preenchidos e $(m-b_0-b_1)$ bits intocados a cada inserção.

A partir destes valores, é possível determinar a fração de bits zerados e a fração de bits preenchidos do vetor depois das n inserções. A probabilidade $p_0(n)$ de um bit estar em 0 após as n inserções é calculada através das probabilidades de $n+1$ eventos mutuamente exclusivos. O primeiro evento é aquele onde o bit está inicialmente em 0 e permanece intocado pelos n elementos inseridos. Notando que $p_0(0)$ representa a probabilidade do bit estar inicialmente em 0, a probabilidade de tal evento é $p_0(0)\,(1-q_0-q_1)^n$. Os outros n eventos são aqueles onde o bit é zerado pelo $(n-i)$-ésimo elemento e não é mais tocado pelos i elementos seguintes, para $0 \leq i \leq n-1$. A probabilidade de ocorrência de cada um destes eventos é expressa por $q_0\,(1-q_0-q_1)^i$. Dessa forma, a probabilidade $p_0(n)$ pode ser expressa por

$$\begin{aligned} p_0(n) &= p_0(0)\,(1-q_0-q_1)^n + \sum_{i=0}^{n-1} q_0\,(1-q_0-q_1)^i \\ &= p_0(0)\,(1-q_0-q_1)^n + q_0\left[\frac{1-(1-q_0-q_1)^n}{1-(1-q_0-q_1)}\right] \\ &= p_0(0)\,(1-q_0-q_1)^n + \frac{q_0}{q_0+q_1}\Big[1-(1-q_0-q_1)^n\Big]. \end{aligned} \quad (18)$$

Analogamente, a probabilidade $p_1(n)$ de um bit estar em 1 depois das n inserções é

$$\begin{aligned} p_1(n) &= p_1(0)\,(1-q_0-q_1)^n + \sum_{i=0}^{n-1} q_1\,(1-q_0-q_1)^i \\ &= p_1(0)\,(1-q_0-q_1)^n + q_1\left[\frac{1-(1-q_0-q_1)^n}{1-(1-q_0-q_1)}\right] \\ &= p_1(0)\,(1-q_0-q_1)^n + \frac{q_1}{q_0+q_1}\Big[1-(1-q_0-q_1)^n\Big], \end{aligned} \quad (19)$$

e obviamente $p_0(n)+p_1(n)=1$. Dado que o mesmo cálculo pode ser aplicado para todos os bits do vetor, na média, uma fração $p_0(n)$ dos bits fica em 0 e uma fração $p_1(n)$ dos bits fica em 1 depois de n inserções.

A partir das proporções de bits no vetor, a probabilidade de falso positivo é calculada de maneira trivial. Dado que na média b_0 bits são zerados e b_1 bits são preenchidos a cada inserção de um elemento, a probabilidade de falso positivo f_p de um FBG é calculada da seguinte forma

$$f_p = p_0(n)^{b_0}\,p_1(n)^{b_1} = p_0(n)^{b_0}\,[1-p_0(n)]^{b_1} = [1-p_1(n)]^{b_0}\,p_1(n)^{b_1}. \quad (20)$$

Falsos Negativos

Falsos positivos ocorrem para elementos externos com a mesma probabilidade para cada elemento checado. Por outro lado, falsos negativos ocorrem somente para elementos *inseridos* com uma probabilidade diferente para cada elemento. Na verdade, um fator que afeta diretamente a probabilidade de falso negativo é a ordem de inserção. Por exemplo, os elementos inseridos primeiro têm uma maior probabilidade de serem falsos negativos do que os elementos inseridos por último. Isso ocorre porque mais inserções são realizadas depois dos primeiros elementos e, portanto, é mais provável que um dos seus bits marcados seja invertido.

A probabilidade de falso negativo pode ser calculada se antes for determinada a probabilidade de um bit do $(n-i)$-ésimo elemento inserido não ser invertido pelos i elementos seguintes, para $0 \leq i \leq n-1$. Como nos falsos positivos, a probabilidade $p_{00}(n-i)$ de um bit zerado pelo $(n-i)$-ésimo elemento permanecer zerado ao fim das i inserções é calculada a partir das probabilidades de $i+1$ eventos mutuamente exclusivos. O primeiro evento é aquele em que o bit permanece intocado por todas as i inserções subseqüentes; tal evento ocorre com probabilidade $(1-q_0-q_1)^i$. Os outros i eventos são aqueles onde o bit é zerado pelo $(n-j)$-ésimo elemento e não é mais tocado pelos j elementos seguintes, para $0 \leq j \leq i-1$. Dessa maneira, a probabilidade $p_{00}(n-i)$ é expressa por

$$\begin{aligned} p_{00}(n-i) &= (1-q_0-q_1)^i + \sum_{j=0}^{i-1} q_0 (1-q_0-q_1)^j \\ &= (1-q_0-q_1)^i + q_0 \left[\frac{1-(1-q_0-q_1)^i}{1-(1-q_0-q_1)} \right] \\ &= (1-q_0-q_1)^i + \frac{q_0}{q_0+q_1} \left[1-(1-q_0-q_1)^i \right]. \end{aligned} \tag{21}$$

Analogamente, a probabilidade $p_{11}(n-i)$ de um bit preenchido pelo $(n-i)$-ésimo elemento permanecer preenchido após as i inserções seguintes é

$$\begin{aligned} p_{11}(n-i) &= (1-q_0-q_1)^i + \sum_{j=0}^{i-1} q_1 (1-q_0-q_1)^j \\ &= (1-q_0-q_1)^i + q_1 \left[\frac{1-(1-q_0-q_1)^i}{1-(1-q_0-q_1)} \right] \\ &= (1-q_0-q_1)^i + \frac{q_1}{q_0+q_1} \left[1-(1-q_0-q_1)^i \right]. \end{aligned} \tag{22}$$

A partir das Equações 21 e 22, a probabilidade de falso negativo de qualquer elemento inserido no filtro pode ser determinada. A probabilidade de falso negativo $f_n(n-i)$ do $(n-i)$-ésimo elemento não ser reconhecido pelo filtro após as i inserções subseqüentes é calculada tomando-se o complemento da probabilidade de nenhum de seus bits serem

invertidos. Logo, esta probabilidade é igual a

$$f_n(n-i) = 1 - p_{00}(n-i)^{b_0} p_{11}(n-i)^{b_1}. \tag{23}$$

O Filtro de Bloom como um Caso Particular

Como esperado, o Filtro de Bloom é na verdade um caso particular do FBG. Para os falsos positivos, a Equação 20 se reduz à probabilidade do filtro convencional quando seus parâmetros são usados, isto é, $k_0 = 0$, $p_0(0) = 1$ e $p_1(0) = 0$. Nesse caso, tem-se $p_0(n) = e^{-k_1 n/m}$, $p_1(n) = 1 - e^{-k_1 n/m}$, $b_0 = 0$ e $b_1 = m(1 - e^{-k_1/m})$. Expandindo a expressão de b_1 em série e notando que $m \gg k_1$, pode-se chegar à simplificação realizada para o caso convencional

$$\begin{aligned} b_1 &= m(1 - e^{-k_1/m}) \\ &= m\left[1 - \left(1 - \frac{k_1}{m} + \frac{k_1^2}{2m^2} - \cdots\right)\right] \approx k_1. \end{aligned} \tag{24}$$

Logo, a probabilidade de falso positivo f_p se reduz à probabilidade do Filtro de Bloom convencional, conforme a Equação 14.

De forma semelhante, a probabilidade de falso negativo na Equação 23 se reduz a zero para o caso do Filtro de Bloom convencional. Nesse caso, $k_0 = 0$ e, portanto, $b_0 = 0$ e $p_{11} = 1$, para $0 \leq i \leq n-1$. Assim, independentemente de outros parâmetros, a probabilidade de falso negativo é zero. Para o último elemento inserido, ou seja, para o n-ésimo elemento, esta probabilidade também é zero dado que nenhum outro elemento pode inverter os seus bits. Neste caso, $i = 0$ e $p_{00} = p_{11} = 1$; logo, a probabilidade de falso negativo também se reduz a zero.

Aplicação

No sistema de rastreamento proposto, os elementos inseridos no FBG são na verdade os endereços IP dos roteadores. Portanto, o número de elementos n representa o número de roteadores atravessados pelo pacote de ataque. Além disso, o tamanho do vetor de bits m é exatamente o número de bits alocados no cabeçalho do pacote para o FBG. Os parâmetros k_0 e k_1 são o número de funções *hash* que zeram e preenchem bits em cada roteador, respectivamente. Estas funções devem ser as mesmas em cada roteador de forma a permitir a reconstrução da rota de ataque posteriormente. Por fim, $p_0(0)$ e $p_1(0)$ representam a fração inicial de bits zerados e a fração inicial de bits preenchidos, respectivamente. Esses dois parâmetros estão sob o controle do atacante, que é responsável pela criação do pacote de ataque e pela inicialização do conteúdo do FBG.

A implementação do Filtro de Bloom nos roteadores muda um pouco com a generalização. Existem bits que são zerados e bits que são preenchidos a cada inserção e, portanto, uma única operação OU bit-a-bit não é suficiente para atualizar o filtro. Dessa forma, é necessária uma operação OU bit-a-bit seguida de uma operação E bit-a-bit para

preencher e zerar os bits indicados, respectivamente. Assim, dois registradores são necessários para cada interface do roteador. Para configurar esses registradores, o endereço IP da interface é usado como entrada para as funções *hash*. O primeiro registrador tem todos os seus bits inicializados com 0 e os bits indicados pelas funções h_j são preenchidos com 1. O segundo registrador tem os seus bits inicializados com 1 e os bits indicados pelas funções g_i são colocados em 0. Quando o pacote vai ser reencaminhado, o FBG do pacote é atualizado inicialmente com o resultado de uma operação OU bit-a-bit do próprio FBG com o primeiro registrador da interface. Em seguida, uma operação E bit-a-bit do FBG com o segundo registrador é realizada para finalizar a atualização.

A ordem das operações é um resultado da prioridade adotada. Na verdade, se for decidido que as funções h_j têm precedência sobre as funções g_i, a única modificação a ser realizada é que a operação E deve ser realizada antes da operação OU.

1.5.2. Um Procedimento Aprimorado de Reconstrução de Rota

Se por um lado o uso de um FBG limita a ação do atacante na geração de falsos positivos, por outro, ele introduz falsos negativos no procedimento de reconstrução de rota. Um falso negativo significa não detectar um roteador por onde o pacote realmente passou. Ao deixar de detectar um roteador atravessado, também não são detectados todos os roteadores cuja rota depende deste roteador. Por exemplo, suponha que na reconstrução ilustrada na Figura 1.22 há um falso negativo no roteador R_4. Ou seja, o roteador R_4 não é reconhecido pelo filtro generalizado durante a verificação realizada por R_2 (3). Neste caso, os roteadores R_3 e R_5 não são checados, uma vez que R_4 não foi integrado à rota de ataque. Em conseqüência, o roteador R_5 também não é integrado à rota de ataque. Desta forma, somente um falso negativo pode ser suficiente para interromper o procedimento de reconstrução e evitar a determinação da verdadeira rota de ataque.

Para resolver este problema, é proposto um procedimento aprimorado de reconstrução que elimina os falsos negativos ao custo de uma maior probabilidade de falso positivo. Este novo procedimento de reconstrução é baseado no fato que, durante a travessia do pacote pela rede, roteadores subseqüentes podem inverter os bits marcados por roteadores anteriores e causar falsos negativos durante a reconstrução. No exemplo anterior, durante o percurso do pacote de ataque por (R_5, R_4, R_2, R_1), ou R_1 ou R_2 inverteu algum bit marcado previamente por R_4. Em conseqüência, o filtro do pacote recebido não reconhece R_4 durante o procedimento de reconstrução de rota. De forma a evitar este problema, cada roteador envia junto com o FBG algumas informações adicionais para os seus vizinhos durante a reconstrução. Esta informação contém os bits marcados pelo próprio roteador e pelos roteadores anteriores no procedimento de reconstrução. A Figura 1.24 ilustra brevemente este conceito. Na figura, o roteador R_1 primeiramente reconhece o roteador R_2 no FBG recebido pela vítima e, portanto, R_1 lhe repassa o FBG de forma a continuar a reconstrução. Entretanto, R_1 também envia outros dois vetores de bits, m_0 e m_1, indicando quais bits foram zerados e preenchidos, respectivamente, por R_1 durante a travessia do pacote. Por sua vez, o roteador R_2, ao reconhecerem algum roteador vizinho, também lhe repassa o próprio FBG e os dois vetores de bits adicionais indicando as marcações feitas por ele e por R_1. Desta forma, um roteador R_i sempre repassa para os próximos roteadores o FBG junto com dois vetores de bits, m_0 e m_1, indicando os bits zerados e preenchidos por R_i e pelos roteadores anteriores no procedimento de reconstrução.

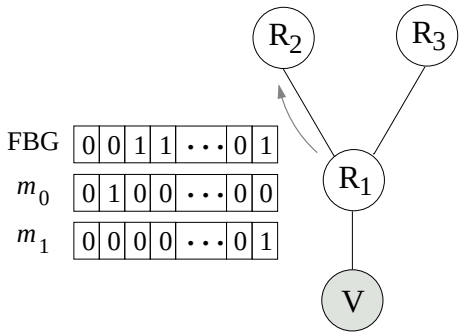

Figura 1.24. O procedimento aprimorado de reconstrução de rota.

O processo de verificação dos vizinhos se altera um pouco em virtude da informação adicional recebida por cada roteador. Primeiramente, antes de checar a presença de cada vizinho no FBG, o roteador atualiza os vetores m_0 e m_1 usando o endereço IP da interface pela qual a requisição de reconstrução chegou. Após a atualização, os testes dos vizinhos são iniciados. Quando um vizinho não é reconhecido pelo FBG, o roteador não descarta aquele vizinho diretamente. Ao invés disso, o roteador verifica se algum bit marcado por este vizinho foi invertido por algum roteador seguinte durante a travessia do pacote. Para isso, ele utiliza os vetores m_0 e m_1 recebidos. Desta forma, caso os bits do vizinho que estão invertidos no FBG estiverem marcados como reescritos, então o vizinho é reconhecido como um elemento do filtro. Caso contrário, aquele vizinho é então descartado. No exemplo dado anteriormente, ao não reconhecer o roteador R_4 no FBG, o roteador R_2 verifica se os bits invertidos de R_4 estão marcados ou no vetor m_0 ou no vetor m_1. Como certamente eles estão, o roteador R_4 é integrado à rota de ataque e o procedimento de reconstrução continua.

Uma importante vantagem do procedimento aprimorado de reconstrução é que falsos negativos não podem mais ocorrer. Dado que os bits reescritos de cada roteador agora podem ser checados a cada salto, os roteadores realmente atravessados são sempre integrados à rota reconstruída. Portanto, a rota de ataque real está *sempre* no grafo de ataque reconstruído. Por outro lado, a probabilidade de falso positivo aumenta à medida que um roteador é testado mais longe da vítima e mais perto do atacante. Isso ocorre porque a fração dos bits marcados em m_0 e m_1 aumenta para cada roteador integrado à rota de ataque. Assim, roteadores que não eram antes reconhecidos porque algum dos seus bits estava invertido, agora podem ser reconhecidos com maior probabilidade. Entretanto, resultados experimentais comprovam que isto não é um problema grave e que o atacante pode ser facilmente identificado [Laufer et al., 2005a], [Laufer, 2005].

A probabilidade de falso positivo do procedimento aprimorado de reconstrução de rota pode ser calculada de uma maneira simples. Primeiramente, é calculada a probabilidade de um bit específico de m_0 estar preenchido em um determinado roteador durante a reconstrução. É importante ressaltar que um bit preenchido em um roteador não significa necessariamente que o bit foi preenchido por este roteador. No caso, o bit pode ter sido preenchido pelo próprio roteador ou por algum roteador anterior no procedimento de reconstrução. Este evento ocorre com probabilidade q_0, uma vez que preencher um bit de m_0 é equivalente a zerar um bit no FBG. Portanto, a probabilidade $s_0(i)$ de um bit

específico de m_0 estar preenchido em um roteador a i saltos da vítima é a probabilidade de pelo menos um dos roteadores anteriores ou o próprio roteador preencher o bit. Desta forma, a probabilidade $s_0(i)$ é

$$s_0(i) = 1 - (1 - q_0)^i. \tag{25}$$

De maneira semelhante, a probabilidade $s_1(i)$ de um bit específico de m_1 estar preenchido em um determinado roteador a i saltos da vítima é a probabilidade de pelo menos um roteador anterior ou o próprio roteador preencher o bit. A probabilidade deste evento é q_1, uma vez que preencher um bit em m_1 é equivalente a preencher um bit no FBG. Portanto, a probabilidade $s_1(i)$ é definida como

$$s_1(i) = 1 - (1 - q_1)^i. \tag{26}$$

Como o mesmo cálculo pode ser realizado para cada bit dos dois vetores, na média, uma fração $s_0(i)$ de bits está preenchida em m_0 e uma fração $s_1(i)$ de bits está preenchida em m_1 durante o procedimento de reconstrução em um roteador a i saltos da vítima.

Desta maneira, um bit é interpretado como zero nos testes de pertinência com os vizinhos se um bit estiver zerado no FBG ou se ele estiver preenchido tanto no FBG e no vetor m_1. Portanto, a fração $t_0(i)$ de bits interpretados como zero nos testes de pertinência realizados por um determinado roteador a i saltos da vítima é

$$t_0(i) = p_0(n) + p_1(n)s_1(i). \tag{27}$$

Analogamente, a fração $t_1(i)$ de bits interpretados como um é

$$t_1(i) = p_1(n) + p_0(n)s_0(i). \tag{28}$$

A partir das Equações (27) e (28), a probabilidade de um falso positivo $f_p(i)$ para um determinado roteador a i saltos da vítima no procedimento aprimorado de reconstrução de rota pode ser calculada e expressa por

$$f_p(i) = t_0(i)^{b_0} t_1(i)^{b_1}, \tag{29}$$

onde b_0 e b_1 são respectivamente o número médio de bits em 0 e o número médio de bits em 1 necessários para um falso positivo ocorra, conforme descrito na Subseção 1.5.1.

Referências

[Aljifri et al., 2003] Aljifri, H., Smets, M. e Pons, A. (2003). IP Traceback using Header Compression. *Computers & Security*, 22(2):136–151.

[Bai et al., 2004] Bai, C., Feng, G. e Wang, G. (2004). Algebraic Geometric Code Based IP Traceback. Em *IEEE International Conference on Performance, Computing, and Communications*, páginas 49–56, Phoenix, AZ, EUA.

[Belenky e Ansari, 2003a] Belenky, A. e Ansari, N. (2003a). Accommodating Fragmentation in Deterministic Packet Marking for IP Traceback. Em *IEEE GLOBECOM 2003 Conference*, páginas 1374–1378, San Francisco, CA, EUA.

[Belenky e Ansari, 2003b] Belenky, A. e Ansari, N. (2003b). IP Traceback With Deterministic Packet Marking. *IEEE Communications Letters*, 7(4):162–164.

[Bellovin et al., 2003] Bellovin, S. M., Leech, M. D. e Taylor, T. (2003). ICMP Traceback Messages. *Internet Draft: draft-ietf-itrace-04.txt*.

[Bloom, 1970] Bloom, B. H. (1970). Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 7(13):442–426.

[Broder e Mitzenmacher, 2003] Broder, A. e Mitzenmacher, M. (2003). Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485–509.

[Burch e Cheswick, 2000] Burch, H. e Cheswick, B. (2000). Tracing Anonymous Packets to their Approximate Source. Em *USENIX LISA'00*, páginas 319–327, Nova Orleans, LA, EUA.

[Büyükkokten, 2005] Büyükkokten, O. (2005). Orkut.com. http://www.orkut.com/.

[CERT, 1996] CERT (1996). *CERT Advisory CA-1996-26 Denial-of-Service Attack via ping*. http://www.cert.org/advisories/CA-1996-26.html.

[CERT, 1997] CERT (1997). *CERT Advisory CA-1997-28 IP Denial-of-Service Attacks*. http://www.cert.org/advisories/CA-1997-28.html.

[CERT, 1998] CERT (1998). *CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks*. http://www.cert.org/advisories/CA-1998-01.html.

[Choi e Dai, 2004] Choi, K. H. e Dai, H. K. (2004). A Marking Scheme Using Huffman Codes for IP Traceback. Em *7th International Symposium on Parallel Architectures, Algorithms and Networks - ISPAN'04*, páginas 421–428, Hong Kong, China.

[Cisco, 2003] Cisco (2003). *Cisco Security Advisory: Cisco IOS Interface Blocked by IPv4 Packets*. Cisco Systems, Inc.
http://www.cisco.com/warp/public/707/cisco-sa-20030717-blocked.shtml.

[CNN.com, 2000] CNN.com (2000). *Denial of service hackers take on new targets*.
http://www.cnn.com/2000/TECH/computing/02/09/denial.of.service.03.

[Dean et al., 2002] Dean, D., Franklin, M. e Stubblefield, A. (2002). An Algebraic Approach to IP Traceback. *ACM Transactions on Information and System Security*, 5(2):119–137.

[Dittrich, 1999a] Dittrich, D. (1999a). *The DoS Project's 'trinoo' distributed denial of service attack tool*. http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt.

[Dittrich, 1999b] Dittrich, D. (1999b). *The 'stacheldraht' distributed denial of service attack tool*. http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt.

[Dittrich, 1999c] Dittrich, D. (1999c). *The 'Tribe Flood Network' distributed denial of service attack tool*. http://staff.washington.edu/dittrich/misc/tfn.analysis.txt.

[Ferguson e Senie, 2000] Ferguson, P. e Senie, D. (2000). Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. *RFC 2827*.

[FTC, 2005] FTC (2005). The CAN-SPAM Act: Requirements for Commercial Emailers. http://www.ftc.gov/bcp/conline/pubs/buspubs/canspam.htm.

[Garber, 2000] Garber, L. (2000). Denial-of-Service Attacks Rip the Internet. *IEEE Computer*, 4(33):12–17.

[Gibson, 2001] Gibson, S. (2001). *The Strange Tale of the Attacks Against GRC.COM*. Gibson Research Corporation. http://www.grc.com/dos/grcdos.htm.

[Globo Online, 2005] Globo Online (2005). Brasil é 5º maior receptor de spam; spywares representam 22% das infecções. http://oglobo.globo.com/online/plantao/169450846.asp.

[Gont, 2004] Gont, F. (2004). ICMP Attacks Against TCP. *Internet Draft: draft-gont-tcpm-icmp-attacks-03.txt*.

[Goodrich, 2002] Goodrich, M. T. (2002). Efficient Packet Marking for LargeScale IP Traceback. Em *9th ACM Conference on Computer and Communications Security - CCS'02*, páginas 117–126, Washington, DC, EUA.

[Gordon et al., 2005] Gordon, L. A., Loeb, M. P., Lucyshyn, W. e Richardson, R. (2005). *2005 CSI/FBI Computer Crime and Security Survey*.

[Gray e Fried, 2003] Gray, P. e Fried, I. (2003). *Al-Jazeera suffers DoS attack*. ZDNet UK. http://news.zdnet.co.uk/business/0,39020645,2132585,00.htm.

[Grupo Brasil AntiSPAM, 2005] Grupo Brasil AntiSPAM (2005). Código de Ética AntiSPAM e Melhores Práticas de Uso de Mensagens Eletrônicas. http://brasilantispam.org/main/codigo.htm.

[Hilgenstieler e Duarte Jr., 2004] Hilgenstieler, E. e Duarte Jr., E. P. (2004). Uma Arquitetura para Rastreamento de Pacotes na Internet. Em *IV Workshop em Segurança de Sistemas Computacionais - WSeg 2004*, Gramado, RS, Brasil.

[Holmes, 2005] Holmes, N. (2005). In Defense of Spam. *IEEE Computer Magazine*, 38(4):86–88.

[Hormel Foods, 2000] Hormel Foods (2000). Your Use of Our Trademark SPAM on Your "Page-O-SPAM" Website. http://www.rsi.com/spam/.

[IBM Report, 2005] IBM Report (2005). Phishing attacks in May jumped more than 200 percent. Relatório técnico, IBM.

[ICQ, 2005] ICQ (2005). Icq.com - community, people search and messaging service! http://www.icq.com/.

[Jacobson, 1990] Jacobson, V. (1990). Compressing TCP/IP Headers for Low-Speed Serial Links. *RFC 1144*.

[Laufer, 2005] Laufer, R. P. (2005). Rastreamento de Pacotes IP contra Ataques de Negação de Serviço. Tese de mestrado, COPPE/UFRJ.

[Laufer et al., 2005a] Laufer, R. P., Velloso, P. B., de O. Cunha, D. e Duarte, O. C. M. B. (2005a). Um Procedimento Alternativo de Reconstrução de Rota para o Rastreamento de Pacotes IP. Em *XXII Simpósio Brasileiro de Telecomunicações - SBrT'05*, Campinas, SP, Brasil.

[Laufer et al., 2005b] Laufer, R. P., Velloso, P. B. e Duarte, O. C. M. B. (2005b). Defeating DoS Attacks with IP Traceback. Em *IFIP Open Conference on Metropolitan Area Networks - MAN'2005*, Ho Chi Minh, Vietnã.

[Laufer et al., 2005c] Laufer, R. P., Velloso, P. B. e Duarte, O. C. M. B. (2005c). Um Novo Sistema de Rastreamento de Pacotes IP contra Ataques de Negação de Serviço. Em *XXIII Simpósio Brasileiro de Redes de Computadores - SBRC'2005*, Fortaleza, CE, Brasil.

[Lee et al., 2004] Lee, T.-H., Wu, W.-K. e Huang, T.-Y. W. (2004). Scalable Packet Digesting Schemes for IP Traceback. Em *IEEE International Conference on Communications ICC'04*, páginas 1008–1013, Paris, França.

[Leech, 1996] Leech, M. (1996). DefUsername/Password Authentication for SOCKS V5. *RFC 1929*.

[Li et al., 2002] Li, J., Mirkovic, J., Wang, M., Reiher, P. e Zhang, L. (2002). SAVE: Source Address Validity Enforcement Protocol. Em *Proceedings of the IEEE INFOCOM 2002 Conference*, páginas 1557–1566, Nova Iorque, NY, EUA.

[Li et al., 2004] Li, J., Sung, M., Xu, J. e Li, L. (2004). Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. Em *Proceedings of the 25th IEEE Symposium on Security and Privacy*, Oakland, CA, EUA.

[Liu et al., 2003] Liu, J., Lee, Z.-J. e Chung, Y.-C. (2003). Efficient Dynamic Probabilistic Packet Marking for IP Traceback. Em *IEEE International Conference on Networks - ICON'03*, páginas 475–480, Sydney, Austrália.

[Mankin et al., 2001] Mankin, A., Massey, D., Wu, C.-L., Wu, S. F. e Zhang, L. (2001). On Design and Evaluation of "Intention-Driven" ICMP Traceback. Em *Proceedings of the IEEE ICCCN 2001 Conference*, Scottsdale, AZ, EUA.

[M.Degermark et al., 1999] M.Degermark, Nordgren, B. e S.Pink (1999). IP Header Compression. *RFC 2507*.

[Microsoft, 2002] Microsoft (2002). *Stop 0A in Tcpip.sys When Receiving Out Of Band (OOB) Data*. Microsoft Corporation. http://support.microsoft.com/kb/q143478.

[Microsoft, 2005] Microsoft (2005). Microsoft Network - MSN. http://messenger.msn.com/.

[Mirkovic et al., 2004] Mirkovic, J., Dietrich, S., Dittrich, D. e Reiher, P. (2004). *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall PTR, 1ª edição.

[Mirkovic e Reiher, 2004] Mirkovic, J. e Reiher, P. (2004). A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communications Review*, 34(2):39–53.

[Mitzenmacher, 2002] Mitzenmacher, M. (2002). Compressed Bloom Filters. *IEEE/ACM Transactions on Networking*, 10(5):604–612.

[Moore et al., 2001] Moore, D., Voelker, G. e Savage, S. (2001). Inferring Internet Denial of Service Activity. Em *Proceedings of the 2001 USENIX Security Symposium*, Washington, DC, EUA.

[Park e Lee, 2001] Park, K. e Lee, H. (2001). On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack. Em *Proceedings of the IEEE INFOCOM 2001 Conference*, Anchorage, AK, EUA.

[Perkins, 2002] Perkins, C. E. (2002). IP Mobility Support for IPv4. *RFC 3220*.

[Postel, 1981] Postel, J. (1981). Internet Protocol. *RFC 791*.

[Postel, 1983] Postel, J. (1983). Character Generator Protocol. *RFC 864*.

[Reuters, 2004] Reuters (2004). *Scotland Yard and the case of the rent-a-zombies*. ZDNet.com. http://news.zdnet.com/2100-1009_22-5260154.html.

[Sahami et al., 1998] Sahami, M., Dumais, S., Heckerman, D. e Horvitz, E. (1998). A bayesian approach to filtering junk E-mail. Em *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, WI, EUA. AAAI Technical Report WS-98-05.

[Savage et al., 2001] Savage, S., Wetherall, D., Karlin, A. e Anderson, T. (2001). Network Support for IP Traceback. *IEEE/ACM Transactions on Networking*, 9(3):226–237.

[Schuba et al., 1997] Schuba, C. L., Krsul, I. V., Kuhn, M. G., Spafford, E. H., Sundaram, A. e Zamboni, D. (1997). Analysis of a Denial of Service Attack on TCP. Em *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, páginas 208–223, Oakland, CA, EUA.

[Shachtman, 2003] Shachtman, N. (2003). *Porn Purveyors Getting Squeezed*. Wired News. http://wired-vig.wired.com/news/print/0,1294,59574,00.html.

[Snoeren et al., 2001] Snoeren, A. C., Partridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Kent, S. T. e Strayer, W. T. (2001). Hash-Based IP Traceback. Em *Proceedings of the ACM SIGCOMM'01 Conference*, páginas 3–14, San Diego, CA, EUA.

[Snoeren et al., 2002] Snoeren, A. C., Partridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Schwartz, B., Kent, S. T. e Strayer, W. T. (2002). Single-Packet IP Traceback. *IEEE/ACM Transactions on Networking*, 10(6):721–734.

[Song e Perrig, 2001] Song, D. X. e Perrig, A. (2001). Advanced and Authenticated Marking Schemes for IP Traceback. Em *Proceedings of the IEEE INFOCOM 2001 Conference*, Anchorage, AK, EUA.

[SpamAssassin, 2005] SpamAssassin (2005). The apache spamassassin project. http://spamassassin.apache.org/.

[Spammer-X et al., 2004] Spammer-X, Posluns, J. e Sjouwerman, S. (2004). *Inside the SPAM Cartel: Trade Secrets from the Dark Side*. Syngress Publishing, 1^a edição.

[Stone, 2000] Stone, R. (2000). CenterTrack: An IP Overlay Network for Tracking DoS Floods. Em *9th USENIX Security Symposium*, páginas 199–212, Denver, CO, EUA.

[US-CERT, 2005] US-CERT (2005). *Vulnerability Note VU#222750: TCP/IP Implementations Do Not Adequately Validate ICMP Error Messages*. http://www.kb.cert.org/vuls/id/222750.

[Wagner, 2002] Wagner, J. (2002). *Dealing With Massive Attack: DNS Protection*. Internetnews.com. http://www.internetnews.com/dev-news/article.php/1487331.

[Watson, 2004] Watson, P. A. (2004). Slipping in the Window: TCP Reset Attacks. Em *2004 CanSecWest Conference*, Vancouver, Canadá.

Capítulo 2

Segurança em Grades Computacionais

José de Ribamar Braga Pinheiro Junior, Fabio Kon
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

Abstract

Computational grids enable the execution of applications in machines geographically distributed across different administrative domains and institutions. The possibility of computers to interact in this context leads to new security problems and increases the complexity of efficient solutions for resource sharing in computational grids. The security requirements for a computational grid differ from those found on more restricted and controlled environments. This chapter introduces the most relevant security aspects for grid computing, describes existing security mechanisms used on grids, and explains how these aspects are addressed by concrete systems.

Resumo

As grades computacionais permitem a execução de aplicações em computadores geograficamente dispersos e pertencentes a instituições e domínios administrativos diferentes. A possibilidade de computadores interagirem neste contexto traz novos problemas de segurança e aumenta a complexidade de soluções eficientes para compartilhamento de recursos em grades computacionais. Os requisitos de segurança para uma grade computacional diferem daqueles presentes em ambientes mais restritos e controlados. Este capítulo descreve os aspectos de segurança mais relevantes para computação em grades, os mecanismos de segurança existentes para tratá-los e a forma como estes aspectos são tratados em sistemas concretos.

2.1. Introdução

A sociedade em que vivemos é cada vez mais dependente das novas tecnologias. A computação, em particular, surgiu para automatizar muitas tarefas que eram executadas de forma arduosa em um tempo não muito distante. A ciência, em todas suas áreas de conhecimento, é sem sombra de dúvida, uma das beneficiadas pelas tecnologias emergentes oriundas da Ciência da Computação. Biólogos, físicos, matemáticos, engenheiros, químicos, por exemplo, utilizam os computadores em suas simulações, otimizações, mineração de dados, entre outras atividades. A indústria, por sua vez, é favorecida, direta ou indiretamente, por resultados conseguidos através da informática para melhorar os seus processos, obtendo maior qualidades em seus produtos e, conseqüentemente, maiores lucros. O comércio, além das vantagens conseguidas naturalmente pela automatização de suas informações, utiliza-se da Internet para aumentar a venda de seus produtos, colocando-os disponíveis diretamente na casa ou no local de trabalho de seus clientes.

Todas essas facilidades fizeram surgir uma crescente necessidade de poder computacional. Problemas já existentes, ou mesmo novos, ainda não experimentados pela sociedade, demandam grande tempo de processamento. Altos investimentos em equipamentos são feitos para resolver problemas computacionais nas áreas de prospecção de petróleo, simulação metereológica, seqüenciamento de DNA e previsão do tempo. A importância estratégica destas atividades nos dão a idéia da importância da computação dentro da sociedade contemporânea. Os altos custos envolvidos na aquisição de computadores de alto desempenho os tornam inacessíveis para uma grande quantidade de potenciais usuários. Por outro lado, os computadores pessoais de hoje possuem a mesma capacidade de processamento dos supercomputadores de uma década atrás. A possibilidade de que estas máquinas disponibilizem seus recursos computacionais para solucionar os problemas de outros usuários e instituições em uma rede de computadores fez surgir a idéia da Computação em Grade (*Grid Computing*)[1] [Foster and Kesselman, 2003, Foster et al., 2001, de Camargo et al., 2004].

Um sistema de Computação em Grade pode ser definido como uma infra-estrutura capaz de interligar e gerenciar diversos recursos computacionais distribuídos por uma rede de computadores de maneira a oferecer, ao usuário, uso intensivo de recursos e aplicações distribuídas [Berman et al., 2003]. Os recursos disponíveis aos usuários das grades são processamento, memória, periféricos (por exemplo, instrumentos científicos e hardware especializado), componentes de software, entre outros.

As redes de computadores, em especial a Internet, no entanto, foram criadas sem a preocupação com a segurança dos dados por elas transmitidos [Garfinkel and Spafford, 1996]. Quando do surgimento das tecnologias de redes de computadores, as relações de segurança entre as partes envolvidas, que eram confiáveis e conhecidas, não obtiveram a preocupação necessária dos seus projetistas. Mais recentemente, a partir do momento que a arquitetura da Internet firmou-se como o padrão de fato das redes de computadores, estas herdaram esse problema e técnicas adicionais tiveram que ser usadas para manter a segurança das informações.

Os computadores que participam de grades computacionais são particularmente

[1] A origem do termo *Grid Computing* é uma analogia com a rede elétrica (*Power Grid*)

vulneráveis a problemas com a segurança. Em um sistema de grade, os recursos a serem protegidos estão localizados em domínios administrativos diferentes, implicando em um controle de acesso mais difícil. A Computação em Grade exige que as soluções de segurança sejam interoperáveis para permitir a integração com os sistemas de segurança já existentes. Finalmente, os sistemas operacionais e o middleware da Grade, por sua vez, podem não ser seguros o bastante, permitindo o acesso indevido a recursos.

Este capítulo foi organizado da seguinte maneira: a Seção 2.2 introduz o problema de segurança, seus fundamentos e requisitos básicos. A Seção 2.3 discute Grades Computacionais, os preceitos de segurança envolvidos e os mecanismos existentes para tratá-los. Na Seção 2.4, apresentamos estudos de casos de implementações de segurança nas principais grades existentes. Ainda nessa seção, as arquiteturas de cada um destes sistemas serão detalhadas. Finalmente, na Seção 2.5 apresentamos as tendências futuras e considerações finais.

2.2. Segurança

A preocupação com a segurança de dados surgiu no meio militar, onde proteger as informações era o principal objetivo, pois delas dependiam os resultados das batalhas. Historicamente sabe-se que muitas das vitórias ocorridas durante a segunda guerra mundial deveu-se à criptografia de dados. Esconder uma ordem que indicava um movimento das tropas poderia fazer uma diferença em centenas de vidas.

Nos dias atuais, os problemas com a proteção dos dados são mais discretos, mas nem por isso menos importantes. Com a grande participação das tecnologias de redes de computadores nas empresas, nas instituições governamentais e até mesmo nos lares de milhões de pessoas, o perigo de uma possível ameaça às informações privilegiadas se faz presente a cada momento. Já faz parte da rotina diária das empresas a realização de negócios utilizando redes privadas e até mesmo a própria Internet como meio de comunicação. Bancos disponibilizam suas transações financeiras através de redes públicas de dados e a interceptação ou modificação de informações podem levar a resultados desastrosos.

A implementação de sistemas seguros deve ser uma preocupação constante nas equipes de desenvolvimento de sistemas computacionais. Um sistema computacional é seguro se pudermos depender dele e seu software tiver um comportamento esperado [Grafinkel and Spafford, 1996]. Sabemos, no entanto, que criar um sistema computacional totalmente seguro é muito difícil (se não impossível) e devemos definir os limites de risco aceitáveis. Respondendo a questões como: qual o perímetro da rede a ser considerada, quais serviços serão disponibilizados externamente, quais são os controles necessários para o sistema e o nível de segurança pretendido, estaremos delineando o escopo do nosso problema.

Cada sistema computacional tem valores associados aos recursos que devem ser protegidos, alguns tangíveis, outros intangíveis [Lang and Schreiner, 2002]. Recursos tangíveis são aqueles aos quais podemos associar valores diretos, ou seja, podemos quantificar um preço por ele (o hardware, por exemplo). Recursos intangíveis (uma informação, por exemplo) são mais difíceis de avaliar pela dificuldade que temos em definir o quanto vale a informação. Lang [Lang and Schreiner, 2002] sugere que devemos quantificar o custo da perda pois é mais apropriado quantificar o impacto negativo do recurso

comprometido: custo da troca, danos à reputação, perda de competitividade, entre outros.

A relação entre o custo da implementação e o custo da perda do recurso nos proporciona o que chamamos de análise de risco. O objetivo da segurança é minimizar o risco. Os custos da implementação de segurança tais como custo com pessoal, hardware, software e tempo gasto não devem ser maior do que o custo associado ao risco potencial de cada possível ataque. É fácil imaginar que os requisitos de segurança necessários para um ambiente bancário devem ser diferentes daqueles esperados para um ambiente educacional. Encontrar um ponto de equilíbrio entre os custos associados à implementação de segurança e o benefício causado por essa implementação é um dos grandes desafios durante o projeto de sistemas de segurança. A seguir trataremos dos objetivos, das políticas e dos mecanismos disponíveis em um sistema de segurança.

2.2.1. Objetivos, políticas e mecanismos

Quando se projeta sistemas de segurança temos que estar atentos aos objetivos a serem alcançados. Os objetivos são definidos pelas políticas e os métodos utilizados para alcançá-los são chamados de mecanismos de segurança [Sinha, 1996].

A política de segurança define as regras gerais pelas quais é fornecido acesso aos recursos. A política serve para avaliar a segurança de um sistema e definir regras e práticas que regulamentam como devemos proteger e distribuir os recursos. As políticas de segurança não devem conter detalhes técnicos relacionados às suas proposições, mas sim especificar o que deve ser feito e o motivo. Escreve-se esta política de forma amigável e com uma linguagem clara e precisa. Uma política sempre é específica para um determinado ambiente e não pode ser definida de forma generalizada.

Os mecanismos de segurança são os meios através dos quais garantimos que uma determinada política está sendo cumprida; os mecanismos são as ferramentas disponíveis para implementar uma política de segurança. Os mecanismos de segurança podem ser desde procedimentos físicos (como impedir a entrada de usuários em uma determinada sala) ou implementados em hardware ou software (como algoritmos de criptografia ou controle de acesso).

Os objetivos comuns de um sistema de segurança são os seguintes:

- Autenticação

  A autenticação é o processo de estabelecer a validade de uma identidade reivindicada [Ramachandran, 2002]. A autenticação é o primeiro passo na segurança de um sistema computacional; junto à confidencialidade e à integridade, consiste num dos pilares da segurança.

- Não Repudiação

  A Não Repudiação consite em obter provas (ou fortes indícios) de ações realizadas no passado de forma que um indivíduo não possa negar ações que tenha realizada no sistema.

- Confidencialidade

A confidencialidade impede que os dados sejam lidos ou copiados por usuários que não possuem o direito de fazê-lo.

- Integridade de Dados

  A integridade de dados protege a informação de ser removida ou alterada sem a autorização do dono.

- Disponibilidade

  A disponibilidade é a proteção dos serviços para que eles não sejam degradados ou fiquem indisponíveis sem autorização. Isto implica em dados e sistemas prontamente disponíveis e confiáveis.

- Controle

  Um sistema computacional pode possuir diversos recursos. O controle permite que somente usuários conhecidos e que têm direitos de acesso em períodos determinados possam, devidamente, dispor dos recursos.

- Prevenção

  A prevenção é um dos elementos fundamentais em todo sistema de segurança. Garfinkel [Grafinkel and Spafford, 1996] diz que a segurança de computadores consiste em uma série de soluções técnicas para problemas não técnicos. Nós podemos gastar grandes quantias de dinheiro, tempo e esforço em sistemas de segurança mas nunca resolveremos problemas como os defeitos desconhecidos nos softwares (*bugs*) ou funcionário maliciosos. Precaver-se em relação a possíveis problemas é uma boa prática de segurança.

- Auditoria

  Ainda não se conhece um sistema de segurança perfeito, sempre é possível que usuários não autorizados possam tentar acessar o sistema, usuários legitimo tenham efetuado ações erradas ou até mesmo atos maliciosos possam ser praticados. Nesses casos é necessário determinar o que aconteceu, quando, quem foi o responsável e o que foi afetado pela ação. A auditoria deve ser um registro incorruptível de principais eventos de segurança. Através da auditoria podemos nos resguardar das ações dos usuários e até mesmo utilizar mecanismos que impossibilitem que os usuários neguem os seus atos ao utilizar o sistema (não repudiação).

2.2.2. Ameaças, vulnerabilidades e ataques

Uma ameaça pode ser definida como um possível perigo que pode explorar uma vulnerabilidade do sistema computacional [Lang and Schreiner, 2002]. Exemplos de ameaça para um sistema computacional são:

- Comprometimento da informação

  Divulgação deliberada ou intencional de informação.

- Violação de integridade

  Modificação maliciosa ou negligente ou destruição de dados.

- Negação de serviço

  Degradação ou impossibilidade de uso de um serviço.

- Repudiação de uma ação

  Falha em verificar a autenticidade de um usuário e prover um método de registrar este fato.

Uma vulnerabilidade é um ponto onde o sistema é susceptível a um ataque. Uma ameaça pode explorar uma vulnerabilidade para concretizar um ataque a um sistema. Uma vulnerabilidade geralmente surge de forma não intencional; por exemplo, para aumentar o desempenho é comum programadores evitarem a verificação de dados de entrada em programas, o que pode levar a vulnerabilidades. Um exemplo comum de vulnerabilidade é o estouro de *buffer* que tem sido muito explorado nos últimos anos [Lhee and Chapin, 2003]. Esta vulnerabilidade consiste em utilizar a falta de verificação do tamanho e do tipo dos dados de entrada de um programa, alterando o seu estado através da inserção de código malicioso, resultando num desvio para executar o código inserido.

O ataque é a efetivação de uma vulnerabilidade através de uma ameaça. O termo intruso ou atacante é comumente usado para representar aquele que tenta obter um acesso não autorizado. Os ataques podem ser passivos ou ativos [Sinha, 1996]. Os ataques passivos ocorrem quando um intruso tenta obter informações de um sistema computacional sem interferir diretamente no funcionamento deste sistema. Os ataques ativos interferem diretamente no funcionamento normal do sistema, geralmente ocasionando algum prejuízo.

O representante mais expressivo dos ataques ativos é conhecido como vírus de computador. Fred Cohen [Cohen, 1987], um dos primeiros pesquisadores que estudou o vírus de computador, o definiu inicialmente como um programa que infecta outro, modificando-o para incluir-se. Devido à facilidade de troca de arquivos pela rede, os vírus passaram a ser um problema constante de segurança. Através da rede, a contaminação por vírus de computador não mais depende do contato direto de dispositivos de armazenamento dos programas. Os vírus atuais propagam-se com velocidade extrema, através dos diversos serviços oferecidos pela Internet e suas vulnerabilidades, dentre os quais destacam-se o correio eletrônico e a Web.

Os dados, ao trafegarem pela rede, podem sofrer ataques do tipo interceptação, interrupção, modificação ou fabricação [Stallings, 2002], como pode ser visto na Figura 2.1. Na interceptação, os dados são copiados e assim perdem sua confidencialidade. A interrupção ocorre quando os dados são perdidos ou corrompidos. Na modificação, os dados são obtidos, modificados e então reenviados ao seu destino. Finalmente, na fabricação, dados e atividades são criados sem sequer terem saído da sua verdadeira origem.

2.2.3. Criptografia

A criptografia tem como objetivo esconder informações sigilosas de qualquer pessoa que não seja autorizadas a lê-las [Terada, 2000]. A criptologia é o estudo de

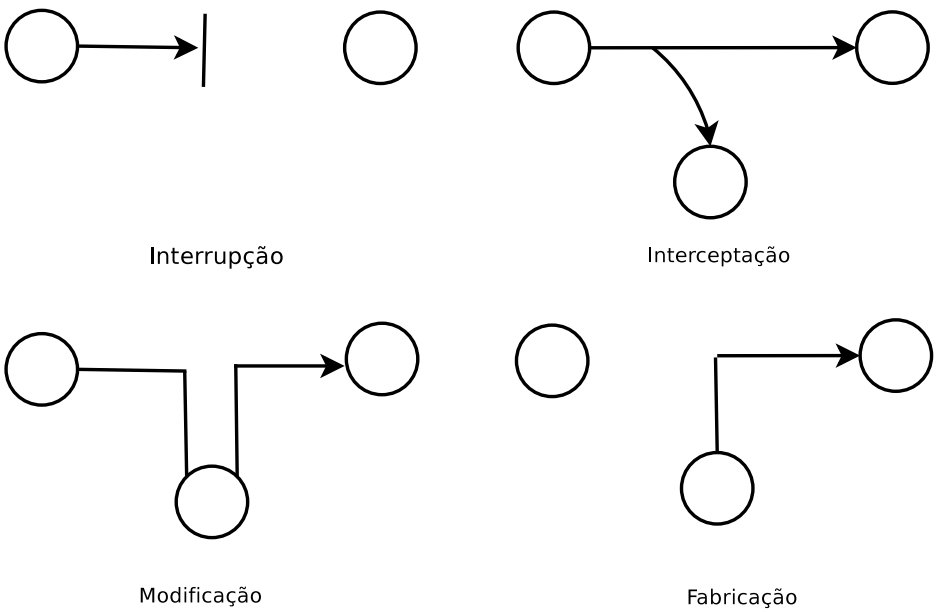

Figura 2.1. ipos de ataques [Stallings, 2002].

técnicas matemáticas relacionadas com a segurança da informação tais como confidencialidade, integridade dos dados, autenticação das entidades e autenticação da origem [Menezes et al., 1996]. A criptografia é então um mecanismo que ajuda a implementar os principais objetivos de segurança de um sistema computacional.

Historicamente a criptografia vem sendo usada pelo ser humano a mais de 2000 anos [Terada, 2000]. A Cifra de César, por exemplo, foi usada pelo Império Romano para esconder informações que eram transmitidas por mensageiros. A cifra de César consistia em deslocar de forma circular 3 posições as letras do alfabeto.

Na Figura 2.2 podemos visualizar um exemplo de uma tabela utilizada para cifrar uma mensagem (cifra de substituição). A primeira linha é o alfabeto na sua seqüência original, enquanto a segunda representa a permutação das letras. Dessa forma duas pessoas que compartilhassem esta tabela poderiam saber que a mensagem "DQJBCM"significa na verdade "GRADES". O ato de procurar a letra trocada na tabela para embaralhar a mensagem é conhecida como encripitar, enquanto a ação contrária correspondente é conhecida como desencriptar. A informação que é compartilhada entre os dois participantes (a tabela) é conhecida como chave criptográfica.

2.2.3.1. Algoritmos criptográficos

Os algoritmos criptográficos são responsáveis pela implementação dos mecanismos de segurança que tornam possível alcançar a maioria dos objetivos de sum sistema de segurança. Existem dois tipos de algoritmos de criptografia: os de chave secreta e os de chave pública (também conhecidos como simétricos e assimétricos, respectivamente). Os algoritmos de chave secreta utilizam uma chave que é compartilhada pelos participantes. Nesse tipo de algoritmo existe o problema de como fazer para compartilhar a chave sem

A	B	C	D	E	F	G	H	I	J	k	L	M
J	A	I	B	C	G	D	F	H	E	N	X	Z

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
R	k	S	L	Q	M	W	Y	P	U	O	V	T

Figura 2.2. Tabela de substituição para cifragem.

que ela fique exposta a intrusos. Nos algoritmos de chave pública cada participante possui um par de chaves: uma privada, conhecida somemente pelo detentor, e outra pública, que pode ser distribuída livremente.

A Figura 2.3 mostra como funciona um esquema de chave secreta. As duas partes, "A" e "B", compartilham uma chave secreta K e desejam se comunicar de maneira que a informação a ser transmitida seja protegida. "A" utiliza a chave que possui e um algoritmo de criptografia para cifrar[2] a mensagem e envia a mensagem no meio de transmissão que estiver disponível. Ao receber o texto cifrado, "B", com a chave que compartilha com seu par, utiliza uma função inversa a usada por "A" e obtém o texto descriptografado, igual ao original.

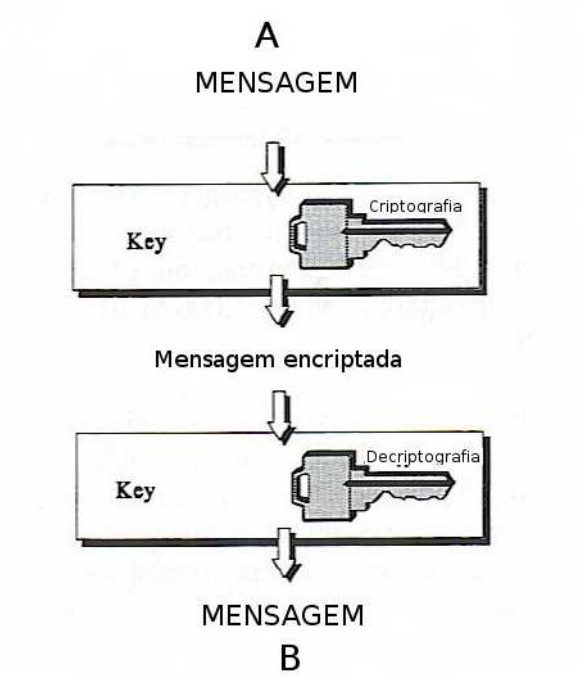

Figura 2.3. Modelo simplificado de chave secreta [NIST SP-800-12, 1995].

Os algoritmos de chave pública foram propostos por Diffie e Hellman em seu

[2]O verbo cifrar é usado no sentido de escrever em cifra, esconder.

famoso artigo *New Directions in cryptography* [Diffie and Hellman, 1976]. No modelo de chave pública, cada usuário possui um par de chaves (S,P), sendo S uma chave secreta e P uma chave pública [Terada, 2000]. Essas duas chaves são relacionadas matematicamente de tal forma que as seguintes propriedades são verdadeiras:

1. S(x) denota a aplicação da chave S no texto x e P(y) = x, ou seja, P é a função inversa de S;
2. O cálculo de S e P é computacionalmente fácil;
3. É computacionalmente difícil calcular S a partir de P;
4. Os cálculos de P() e S() são computacionalmente fáceis;
5. É computacionalmente difícil calcular S() sem conhecer a chave S.

A Figura 2.4 representa a utilização de um algoritmo de criptografia de chave pública. "A" deseja enviar uma mensagem criptografada para "B"; para fazer isso "A" utiliza a chave pública de "B" e transmite a mensagem. "B", quando recebe a mensagem de "A", é o único que consegue recuperar a mensagem por possuir a chave privada correspondente.

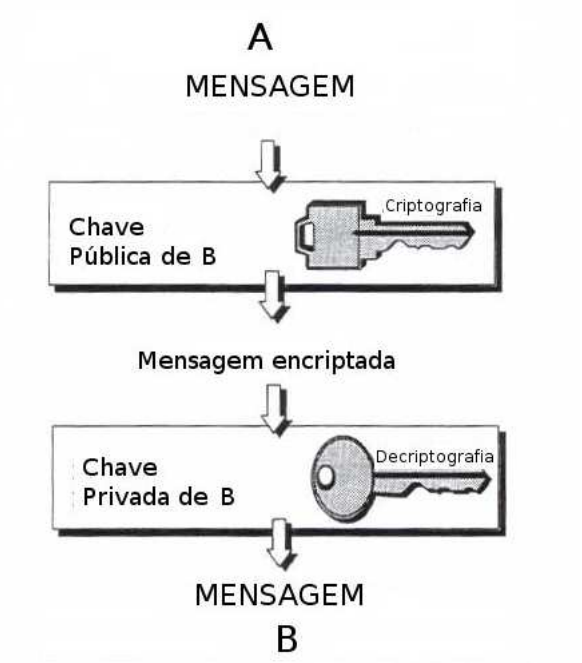

Figura 2.4. Modelo simplificado de chave pública [NIST SP-800-12, 1995].

2.2.3.2. Assinatura digital

Uma assinatura digital é mecanismo criptográfico que tem função similar a uma assinatura escrita à mão [NIST SP-800-12, 1995]. A assinatura possui duas funções primordiais à integridade e a autenticação da mensagem. Vamos considerar que dois usuários

desejam se comunicar utilizando um meio de comunicação de acesso público. A integridade impediria que um usuário mal intencionado alterasse parte da mensagem antes de ela chegar ao seu verdadeiro destino. A autenticação garantiria que a mensagem enviada tenha sido gerada verdadeiramente pelo usuário legítimo. Uma terceira função da assinatura seria a não-repudiação. A não-repudiação permite que, depois da assinatura de uma mensagem, um usuário negue que a tenha feito tal ação.

A Figura 2.5 mostra um processo de assinatura utilizando chaves públicas. Inicialmente o usuário "A" usa sua chave privada para assinar a mensagem a ser enviada para "B". Uma vez recebida a mensagem, "B" utiliza a chave pública de "A" para verificar sua origem. Para diminuir o tempo necessário para a execução do processo é possível utilizar uma função de espalhamento[3] na mensagem e logo depois aplicar o algoritmo de criptografia sobre essa saída, executando o processo inverso no destinatário.

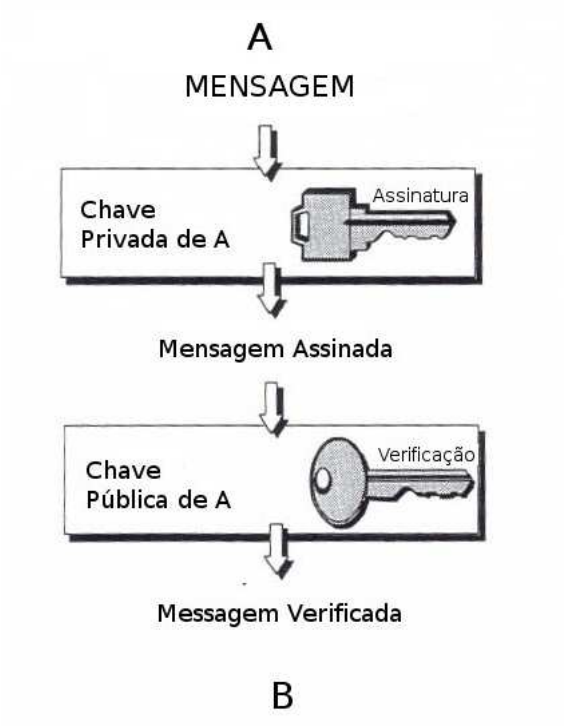

Figura 2.5. Assinatura através de uma chave pública.

2.2.3.3. Certificados digitais

Os certificados digitais são utilizados para garantir que uma determinada chave pública pertença ao seu suposto proprietário. Um certificado é um tipo especial de assinatura digital, ele é um documento que contém a chave pública de um determinado usuário e outras informações adicionais. Este documento é assinado por um terceiro elemento, uma

[3]Uma função de espalhamento (ou *hashing* é uma função matemática que gera uma saída de tamanho fixo e que possui algumas propriedades interessantes entre elas o fato de ser impraticável determinar a entrada a partir de seu hash.

autoridade confiável, atestando a autenticidade da identidade de um usuário. Na Seção 2.3.2.3 trataremos do X.509, um padrão de certificados digitais.

2.2.4. Auditoria

A auditoria é um mecanismo de segurança que tem como objetivo o registro e a análise de atividades relevantes de segurança. A auditoria pode ajudar na detecção de violações de segurança, análise de desempenho ou falhas em aplicações. Manter um registro de eventos é importante pois não é possível garantir se uma determinada política de segurança está correta ou até mesmo se foi implementada corretamente através dos mecanismos de segurança disponíveis.

A auditoria pode fornecer benefícios como o acompanhamento de ações de um usuário, reconstrução de eventos e análise de problemas. Com o acompanhamento de um usuário (ou um processo), as ações ficam associadas com quem as executou, de forma que passa a ser possível realizar uma prestação de contas (não repudiação, por exemplo). Através da reconstrução de eventos, podemos verificar em quais circunstância uma situação relevante ocorreu e quem foi o verdadeiro responsável para que ela tenha ocorrido. Uma análise minuciosa dos problemas de segurança ocorridos facilita a manutenção do sistema de segurança, permitindo que políticas, mecanismos de segurança, o software e o hardware sejam revisados de forma pró-ativa ou preventiva.

A escolha correta dos eventos que deverão ser registrados é um passo importante em um projeto de sistemas de segurança. Escolher o registro de mais eventos do que seria necessário tem um custo computacional muito alto, influenciando no desempenho do sistema como um todo e no uso de recursos. Caso a quantidade de eventos escolhidos seja insuficiente, os registros dos eventos podem ser considerados ineficazes em trazer informações que possam ser utilizadas.

A auditoria é um dos alvos preferenciais dos usuários que tentam atacar um sistema. Paralisando, destruindo ou compremetendo a integridade do sistema de auditoria, o atacante dificulta ou impossibilita ao administrador do sistema que ele e/ou seus atos indevidos sejam descobertos. Políticas de segurança como o uso de mecanismos de criptografia, cópia rotineira dos arquivos de eventos e proteção física das máquinas devem ser consideradas sempre que um sistema de segurança for implementado e isso for relevante.

A seguir apresentamos as Grades Computacionais, seus conceitos básicos, requisitos de segurança e mecanismos disponíveis para implementar segurança em Grades.

2.3. Grades Computacionais

Hoje em dia, apesar de todo avanço da computação nos últimos anos, ainda existe uma necessidade desproporcional de recursos computacionais para resolver alguns problemas específicos em diversas áreas. O custo de máquinas poderosas que possuam os recursos necessários para resolver estes problemas ainda é demasiadamente alto, particulamente para intituições de pequeno e médio porte. As grades computacionais surgiram para tentar dar alento a esta questão; elas são sistemas distribuídos que permitem escalonar processos em máquinas espalhadas por diversas organizações e domínios administrativos.

As grades computacionais estão atualmente em evidência; diversos pesquisadores no mundo todo, nas mais diversas áreas da Computação, investem recursos humanos e financeiros em pesquisas sobre o tema. A indústria já começa a investir no desenvolvimento de sistemas que viabilizem este modelo. Grandes empresas como IBM, Microsoft, NEC, Sun e Oracle investem tempo e dinheiro para desenvolver padrões e sistemas de Grades Computacionais. Algumas questões ainda necessitam de muito trabalho para considerarmos que o desenvolvimento de sistemas de grades esteja totalmente consolidado, sobretudo a Segurança. Entre as forças relacionadas ao desenvolvimento de Grades computacionais [de Camargo et al., 2004] podemos destacar:

- Executar em diversas plataformas sobre os diversos sistemas operacionais pré-existentes, não exigindo assim nenhum tipo de substituição;
- permitir o compartilhamento eficiente de recursos computacionais, tais como processador, memória, disco, outros tipos de hardware e software;
- oferecer qualidade de serviço tanto aos usuários das aplicações quanto aos proprietários dos recursos;
- prover suporte a aplicações paralelas;
- oferecer autonomia para usuários e administradores, evitando impor políticas rígidas de utilização e oferecimento dos recursos;
- ser escalável, podendo englobar de poucas máquinas até possivelmente milhões;
- possuir baixo custo de instalação;
- não causar sobrecarga perceptível ao usuário; e
- ser fácil de usar, permitindo que aplicações pré-existentes sejam facilmente adaptadas ao novo contexto.

O padrão arquitetural *Grid* [de Camargo et al., 2004] apresenta uma arquitetura para grades computacionais (Figura 2.6). Este padrão arquitetural tem como usos conhecidos os sistemas Globus [Foster and Kesselman, 1997], Condor [Epema et al., 1996], InteGrade [Goldchleger et al., 2003] e OurGrid [Andrade et al., 2003] descritos na seção a seguir. Nesse padrão, o agente de acesso é o ponto de acesso primário do usuário à Grade e deve ser executado em cada nó de onde os usuários submeterão aplicações à Grade. Através dele o usuário pode, ao submeter aplicações, definir seus requisitos, monitorar e coletar os resultados das execuções. O serviço de provisão de recursos deve ser executado em cada nó que disponibiliza seus recursos para a Grade; ele gerencia e provê informações sobre a disponibilidades destes recursos. O serviço de provisão de recursos é responsável por permitir a execução de aplicações no nó, reportar erros e devolver os resultados do processamento ao usuário ou a outros nós da aplicação distribuída. O serviço de monitoramento de recursos é responsável por monitorar os estados dos recursos e prover informação sobre o estado destes recursos a outros serviços da Grade. O serviço de escalonamento gerencia os recursos disponíveis na Grade e é responsável, baseado nas informações recebidas do serviço de monitoramento, por escolher nós aptos a executar uma aplicação cuja execução foi solicitada.

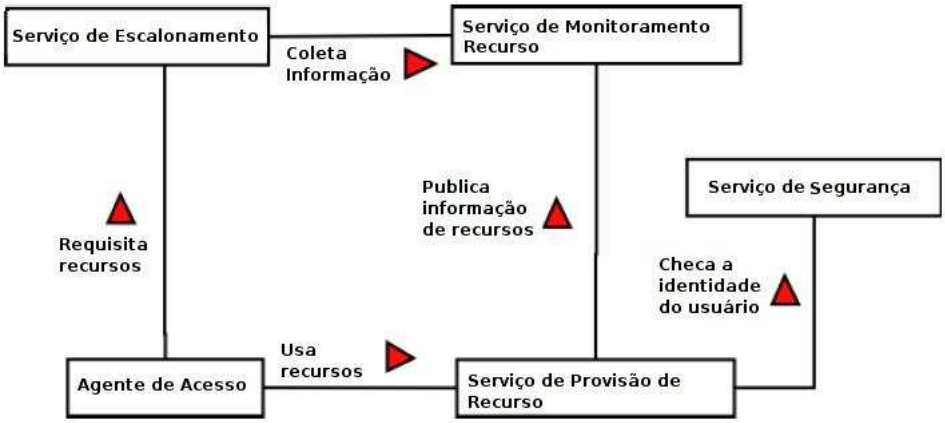

Figura 2.6. Arquitetura de uma Grade computacional.

O serviço de segurança é responsável por três tarefas principais:

1. Proteger os recursos compartilhados, de forma que os usuários que compartilham seus recursos na Grade não sofram os efeitos de aplicações maliciosas.
2. Gerenciar as identidades dos usuários, permitindo relações de confiança e responsabilidade.
3. Proteger as trocas de informações entre os participante da Grade, provendo confidencialidade e integridade.

As grades computacionais ainda apresentam alguns desafios para serem resolvidos. A Segurança em Grades Computacionais, sobretudo, é um tema relevante dentro do contexto anteriormente delineado. Nas subseções a seguir apresentamos os requisitos e os mecanismos de segurança disponíveis relevantes às grades.

2.3.1. Requisitos de segurança

Sistemas de Grades Computacionais são inerentemente mais vulneráveis a ameaças de segurança do que sistemas tradicionais. Em uma grade existe, potencialmente, uma quantidade grande de usuários, recursos e aplicações sendo administradas por diferentes domínios administrativos. Neste ambiente, as regras de segurança definidas em cada sítio podem ser significantemente diferentes. As políticas definidas para um usuário local podem ser ineficientes em um ambiente de Grade, seja porque são permissivas em excesso ou porque são proibitivas demais, forçando aos participante um isolamento pouco produtivo.

As organizações que participam de uma grade computacional geralmente possuem investimentos em sistemas de segurança já existentes [Foster and Kesselman, 2003]. Em uma grade pode existir a necessidade da convivência de vários protocolos de segurança. Os mecanismos diferentes podem exigir interoperabilidade segura entre máquinas e ambientes heterogêneos.

A forma de autenticação e autorização em uma Grade é um requisito importante na definição de um sistema de segurança para a Grade. Pelos requisitos definidos anteriormente, a autenticação na Grade deve permitir a interoperabilidade entre diversos

mecanismos de segurança; isso implica que temos que transformar um mecanismo de segurança em operação de autenticação [Foster and Kesselman, 2003]. Da mesma forma, a autorização poderia fornecer suporte a diversos mecanismos de controle de acesso.

A possibilidade de delegação de direitos de acesso aos elementos de uma Grade é um requisito importante a ser definido. A delegação ameniza o trabalho do administrador em ter que ceder direitos a membros da Grade. Sempre que um dono de recurso confiar em alguém para usar este recurso, este poderá delegar este direito, por um tempo determinado, a um terceiro desde que o sistema assim permita. A delegação de direitos, no entanto, deve ser usada de uma forma restrita. Uma corrente de redelegação muito grande pode fazer com que o recurso acabe sendo usado indevidamente por usuários maliciosos. Os sistemas de segurança de grades devem conter dispositivos para minimizar este problema.

No ambiente de grades surge um problema que não é comum em ambiente tradicionais: devemos garantir que um recurso não seja provido por um atacante. Um usuário mal intencionado, por exemplo, poderia disponibilizar recursos na Grade com a intuito de obter informações privilegiadas. Uma boa política de autorização diminui o risco deste problema acontecer. Em geral o "princípio do menor privilégio" é uma boa opção para a definição de políticas de segurança na Grade. Ou seja, não convém dar ao usuário mais direitos do que ele necessita, mesmo que isso gere um aumento no custo da administração do sistema.

A auditoria é um dos requisitos importantes a ser considerado em uma grade. A diversidade de usuários e de recursos aumenta a probabilidade de surgirem ameaças; eliminar as vulnerabilidades de um sistema nem sempre é possível, principalmente em uma ambiente dinâmico. Os administradores da Grade devem ter mecanismos de rastreamento que permitam verificar ameaças ou erros.

A confidencialidade e a integridade são requisitos importantes para as grades computacionais. Possuir mecanismos que impeçam, ou pelo menos dificultem muito, a modificação ou a obtenção de dados é muito importante. Por questão de desempenho, estas possibilidades devem ser opcionais para os membros.

Estes são os requisitos básicos de segurança necessários para uma grade computacional. A seguir apresentamos os mecanismos de segurança disponíveis para implementar estes requisitos.

2.3.2. Mecanismos de segurança

Nesta seção apresentamos os mecanismos de segurança disponíveis para implementar segurança em grades. Estes mecanismos são apropriados para utilização em sistemas distribuídos e sua aplicação no ambiente de grades computacionais é, em geral, possível. Os mecanismos a seguir descritos são usados em sistemas de rede há anos e passaram ao longo do tempo pr várias melhorias. Esses mecanismos, apesar de tudo, possuem falhas que serão apontada em cada subseção.

2.3.2.1. Kerberos

O Kerberos [Kohl and Neuman, 1993] é um protocolo de autenticação em rede que permite a autenticação de usuários e serviços. Este protocolo oferece confidenciabilidade,integridade e assinatura para as mensagens transmitidas sobre a rede. O protocolo foi criado na década de 80 pelo MIT (Massachusetts Institute of Technology) (`www.mit.edu`) e até hoje ainda é bem utilizado em sistemas distribuídos. Durante anos ele vem sendo aprimorado e encontra-se na sua versão 5. O sistema possui uma implementação disponível sobre forma de código aberto (`http://web.mit.edu/kerberos/www/`).

O Kerberos possui uma estrutura de dados associada a chaves criptográficas, denominada *ticket*. O *ticket* é apresentado sempre que houver a necessidade de autenticação em um serviço ou aplicação. O *ticket* é criptografado e contém informação do identificador do cliente e uma chave aleatória gerada pelo Kerberos.

A arquitetura do Kerberos é constituída por três módulos. O servidor de autenticação (AS - *Autentication Server*), o servidor de concessão de *tickets* (TGS - *Ticket Granting Server*) e um banco de dados. O AS é responsável pela autenticação do usuário na rede. O TGS é um servidor que gera credenciais para comunicação entre clientes e servidores de aplicação. O banco de dados é responsável por armazenar as chaves secretas dos usuários e dos serviços que deverão se autenticar junto ao Kerberos.

O Kerberos usa chaves secretas compartilhadas para proteger a comunicação e permitir autenticação de usuários e de servidores de aplicação. Cada usuário possui uma senha para autenticação. Uma chave criptográfica derivada desta senha é armazenada no banco de dados. Da mesma forma, o servidor de aplicação possui uma chave criptográfica para autenticação.

O processo de autenticação, apresentado na Figura 2.7, é descrito a seguir. Inicialmente um cliente envia para o AS uma solicitação de autenticação. O AS responde com uma credencial criptografada com a chave do cliente. A credencial consiste em um *ticket* para o TGS, denominado TGT (*Ticket-Grainting Ticket*), e uma chave de sessão. Uma chave de sessão é uma chave secreta compartilhada entre o Kerberos e um cliente. Este *ticket* é composto pela identidade do cliente e uma cópia da chave da sessão e é assinado com a chave do servidor TGS. Este *ticket* é usado para requisitar credenciais ao TGS.

Uma vez autenticado, o usuário pode solicitar chaves de sessão para servidores de aplicação. De posse da credencial conseguida no processo de autenticação, o cliente solicita ao TGS uma credencial para um determinado servidor de aplicação; o TGS responde com duas mensagens: um *ticket* criptografado com a chave da sessão TGS para o cliente e outro criptografado com a chave secreta do servidor de aplicação (TA e TB, na figura, respectivamente); ambos os *tickets* possuem uma chave idêntica. O cliente envia ao servidor de aplicação a credencial obtida e este responde com uma mensagem criptografada com a chave compartilhada entre o cliente e o servidor de aplicação, o que confirma sua autenticação.

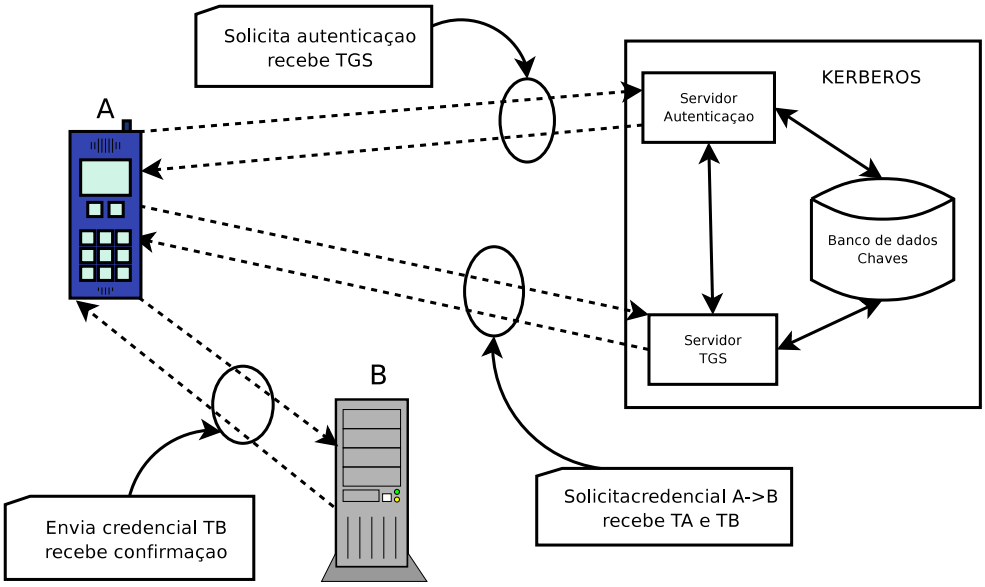

Figura 2.7. Protocolo do Kerberos.

Devido à impossibilidade de todos usuários estarem cadastrados no mesmo servidor, o Kerberos possui a noção de domínios administrativos (*realms* na nomenclatura Kerberos). Esta divisão muitas vezes é feita sobre limites organizacionais. Cada domínio possui sua própria base de dados e seus próprios servidores. Para um usuário utilizar o Kerberos, ele deve estar cadastrado em um domínio. O identificador de um usuário possui a forma NOME_USUÁRIO@DOMÍNIO_USUÁRIO.

Existe também a possibilidade de um usuário de um domínio poder se autenticar em um outro domínio administrado por uma instituição diferente. Para obter chaves de um domínio remoto, o usuário deve solicitar ao seu próprio TGS um *ticket* aceito pelo TGS remoto. Se o TGS remoto tiver seu registro no TGS local, o TGS local entregará ao usuário um *ticket* do domínio remoto. Deste ponto em diante, o usuário usa esse *ticket* para solicitar credenciais para os servidores presentes em outros domínios. Uma outra opção permitida pelo Kerberos é a hierarquia de domínios, de maneira que para contatar um serviço em um outro domínio, pode ser necessário contatar TGSs remotos em um ou mais *realms* imediatos. O nome de cada domínio é armazenado no *ticket*.

Existem algumas opções avançadas disponíveis no protocolo Kerberos através de opções presentes no ticket. A seguir uma breve descrição de cada uma delas.

- INITIAL

  Indica que um *ticket* foi criado usando o protocolo do AS e não via TGS. No Kerberos é possível um cliente solicitar uma credencial a um servidor diretamente do AS. Isso às vezes é interessante para alguns tipos de aplicação de modo a garantir um tempo curto entre a autenticação e a utilização do recurso.

- INVALID

  Indica que o *ticket* é inválido. Servidores de aplicação devem rejeitar este *ticket*. Estes *tickets* podem se tornar válidos através de uma solicitação de validação de *ticket* ao TGS.

- RENEWABLE

  Os *tickets* no Kerberos, por questão de segurança, possuem tempo de vida limitado (cerca de 8 horas). Um *ticket* com este sinal ativado pode ser renovado pelas aplicações (desde que ele já não tenha expirado). Ele possui dois tempos de expiração, quando o *ticket* deve expirar e o tempo a partir do qual ele não mais pode ser renovado.

- POSTDATED

  Um sistema em lote que deve executar em um horário pré-determinado pode necessitar de um **ticket** para uso futuro.

- PROXYABLE

  Indica que o cliente pode permitir que um determinado serviço se comporte como ele. Para limitar o mau uso deste **ticket**, somente endereços de rede listados podem apresentar um *ticket* deste tipo.

- FORWARDABLE

  Esta opção é um caso específico da *PROXYABLE*, a diferença é que, neste caso, o serviço assume a identidade do cliente.

O Kerberos possui algumas limitações bem conhecidas [Bellovin and Merritt, 1990]:

- O Kerberos faz cache de seus *tickets* no diretório temporário do sistema operacional hospedeiro. Ou seja, a segurança do protocolo depende da segurança do sistema do arquivos da máquina onde o Kerberos está sendo executado;
- Como o Kerberos utiliza o tempo para impedir que as mensagens que trafegam na rede sejam retransmitidas ao destinatário (ataques do tipo *replay*), o protocolo exige que as máquinas estejam sincronizadas. Isto pode gerar uma negação de serviço para alguns clientes legítimos;
- As aplicações podem ser modificadas e usadas para capturar senhas dos usuários.

2.3.2.2. SESAME

O SESAME [Sesame, 2003] (*Secure European System for Applications in a Multi-vendor Environment*) é um projeto de desenvolvimento e pesquisa europeu que oferece tecnologia de assinatura única com controle de acesso distribuído baseado em papéis e troca de dados criptografados para sistemas distribuídos. O sistema tem sido aprimorado ao longo dos anos e a sua versão corrente do SESAME é a V4.

A arquitetura do SESAME provê as seguintes características:

- Autenticação simples ou mútua;
- Confidencialidade e integridade na comunicação de dados;
- Controle de acesso baseado em papéis;
- Delegação de direitos;
- Serviço de auditoria;
- Suporte a vários domínios.

De forma resumida, o SESAME trabalha da seguinte forma. Inicialmente o usuário se autentica em um servidor de autenticação e recebe um pacote especial de identificação (*token*) que prova sua identidade. O usuário apresenta este *token* a um servidor de atributos de privilégios que disponibiliza um certificado de controle de acesso. Sempre que for necessário acessar um recurso protegido, o usuário apresenta este certificado ao detentor do recurso que toma suas decisões de acordo com os seus atributos de segurança ou outras informações adicionais, como por exemplo uma lista de controle de acesso. O SESAME permite que os direitos de acesso de um determinado usuário possa ser delegado a um terceiro.

A Figura 2.8 ilustra os componentes do SESAME. O SESAME está organizado em três grupos: os componentes do cliente (*Client-side components*), os componentes do servidor de segurança do domínio (*Domain Security components*) e, finalmente, os compontes do servidor (*Server-side components*) [Tanenbaum and Steen, 2002].

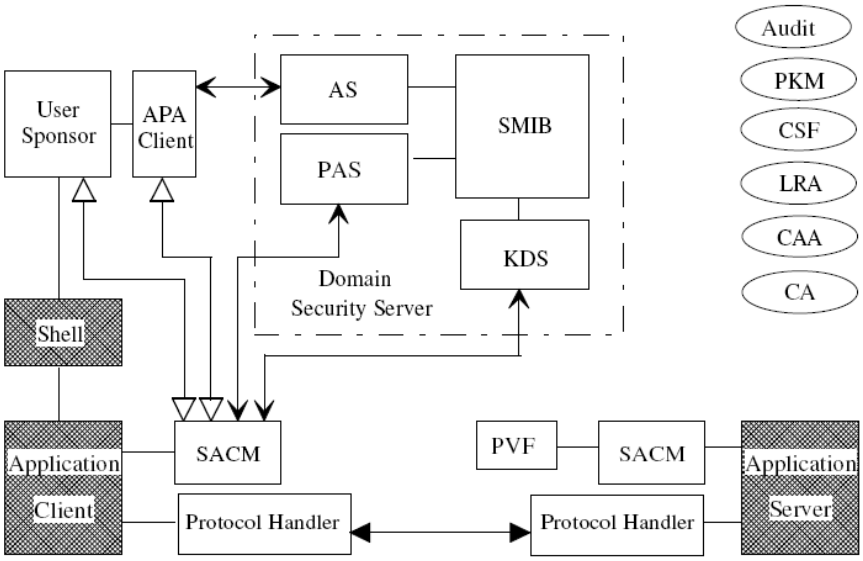

Figura 2.8. Arquitetura do SESAME [Sesame, 2003].

Componentes do servidor de segurança

Os componentes do servidor de segurança são agrupados em uma máquina conhecida como DSS (*Domain Security Server*). Sua função é permitir a autenticação, autorização e distribuição de chaves. O servidor de autenticação (AS - *Authenticator Server*) é responsável pela autenticação de usuários e programas. O SMIB (*Security Menagement Information Base*) é uma base de dados que armazena chaves privadas (das aplicações) e informações relevantes à implementação da segurança no servidor. O PAS (*Privilege Attribute Server*) providencia uma lista de certificados que determinam os direitos dos clientes às aplicações. Ao fazer uma requisição, um usuário recebe um certificado de privilégio (PAC - *Privelege Attribute Certficate*).

O PAC é o elemento central para o SESAME. Ele possui informações sobre uma sessão em particular e é assinado pelo PAS. O PAC permite a delegação temporária dos seus direitos a um outro usuário ou servidor. A delegação de um PAC permite também que uma determinada entidade seja anônima (ou melhor, pseudo-anônima). Para fazer isto, o cliente obtém um PAC delegado que não contém sua identidade e o delega para um terceiro que atua com seus direitos. Desta forma, o elemento final, o servidor, não tem conhecimento sobre a identidade do seu cliente.

Componentes do lado do cliente

No lado do cliente, o SESAME possui os seguintes componentes: o fiador do usuário (*User Sponsor*), o cliente de autenticação e privilégio (*APA Client - Authentication and Privilege Access Client*) e o gerenciador de contexto de segurança (SACM - *Secure Association Context Managemement*). O primeiro permite ao usuário entrar, sair e trocar os seus papéis de acordo com a necessidade de um determinado contexto. O programa que executa a função do fiador do usuário possui uma relação de confiança e interage com o sistema operacional hospedeiro do cliente para implementar algumas de suas funções. Estas funções incluem, por exemplo, gerar chaves secretas a partir de uma senha para comunicações com o servidor do domínio de segurança. O cliente de privilégio e autenticação, implementa uma biblioteca que permite ao usuário e às aplicações interagirem com o servidor de segurança do domínio. E, por último, o gerenciador de contexto da segurança é responsável por iniciar e manter as informações necessárias para a comunicação segura com o servidor de aplicações.

Os privilégios atribuídos aos clientes durante a requisição de um PAC são definidos durante o processo de *logon*. O usuário pode especificar valores padrão associados a um papel que ele assuma ou atributos específicos. No primeiro caso, todos os privilégios associados ao seu identificador e ao seu papel padrão são atribuídos. No caso do cliente indicar somente um papel especifico, tomará somente os privilégios daquele papel. Caso deseje, o cliente pode solicitar somente atributos específicos, o que pode ser interessante no caso de uma delegação.

Componentes do lado do servidor

Assim como o cliente, o servidor também possui um gerenciador de contexto de segurança (SACM), com função análoga. O servidor possui ainda, um componente cuja a função é validar e extrair todas as requisições necessárias das requisições de entrada, o PVF (*PAC Validation Facility*). Assim, por exemplo, o PVF descriptografa chaves de

sessões, extrai os direitos de acessos dos PACs recebidos do cliente e verifica assinaturas.

2.3.2.3. Infra-estrutura de Chaves Públicas

Uma infra-estrutura de chaves públicas (ICP ou PKI - Public Key Infrastructure) reúne um conjunto de hardware, software, políticas, e procedimentos necessários para criar, gerenciar, armazenar, distribuir e revogar certificados de chaves públicas [Adams and Lloyd, 2002]. O ICP foi concebido como um ambiente seguro e eficiente para oferecer serviços de autenticação e segurança baseados nas técnicas de criptografia de chave pública (ver subseção 2.2.3.1).

Entre as tarefas de uma ICP estão o registro de entidades, a inicialização, a certificação, a recuperação e atualização de chaves. Os três primeiras tarefas são conhecidas como matricula da entidade. O registro corresponde ao processo de identificação da entidade, esta tarefa depende das políticas definidas pela ICP. O registro de uma entidade que represente um banco deve ser algo tratado com mais rigor do que a definição de um simples usuário, por exemplo. A inicialização define o mecanismo que será associado a entidade que se registrou, esta ação corresponde etapa da criação das chaves públicas. O processo de inicialização pode ser efetuado pelo ICP ou até mesmo pela própria entidade registrada. A certificação é a conclusão do processo de matricula de uma entidade. Nessa tarefa será criado o Certificado de Chave Pública. O ICP é responsável também pela recuperação e atualização das chaves. A recuperação ocorre quando, por qualquer motivo considerado válido pela ICP, a recuperação da chave for necessária. Os eventos que poderiam originar a perda de chave poderiam ser a quebra de mecanismos de armazenamento, esquecimento, entre outros. Quando um determinado certificado expira é necessário a criação de novas chaves para a entidade. A ICP pode gerar chaves novas e emitir um novo certificado.

Um Certificado de Chave Pública é gerado sempre com um prazo de validade muito grande. Porém, em certas situações, é necessário que um certificado seja considerado não válido e seja revogado. Cabe ao ICP revogar o certificado antes do prazo de expiração e publicar esta informação. O ICP possui uma lista de certificados revogados que é utilizada para verificar um certificado cuja a data de expiração não foi alcançada.

O ICP é composto dos seguintes elementos:

Entidade Final

A Entidade Final corresponde aos elementos que usam a infra-estrutura. As Entidades Finais são representadas pelos usuários, roteadores, servidores, e outros elementos que possam utilizar um certificado de chave pública.

Autoridade Certificadora

A Autoridade Certificadora (AC) é elemento central da infra-estrutura de chaves públicas. Ela é responsável pela emissão de certificados de chaves públicas e de listas de certificados revogados. A Autoridade Certificadora pode, por questões administrativas, delegar muitas das suas tarefas a outros elementos da arquitetura como a Autoridade de Registro ou um Emissor de Lista de Certificados Revogados.

Autoridade de Registro

A Autoridade de Registro (AR) é uma interface entre a Autoridade Certificadora e o usuário do ICP. Essa entidade é opcional na arquitetura, podendo ser usado para a recepção de pedidos de registros de um Entidade Final, verificação da autenticidade dos requerentes, entre outras. A Autoridade de Registro, no entanto não pode emitir um certificado de chave pública, pois esta tarefa é exclusiva da Autoridade de Registro. A opção de delegar algumas tarefas da AC para a AR pode ser interessante por questões administrativas e de segurança. Uma empresa pode, por exemplo, terceirizar as tarefas administrativas de uma ICP com restrições como a impossibilidade de registro de Entidades Finais.

Emissor de Lista de Certificados Revogados

O Emissor de Lista de Certificados é responsável pela emissão e controle das Listas de Certificado Revogados. Este componente é opcional e recebe a delegação de suas tarefas da Autoridade de Registro.

Repositórios de Certificados

O Repositório de Certificados é responsável por armazenar e recuperar certificados e lista de certificados revogados. Um Repositório de Certificados refere-se a qualquer método de armazenamento e recuperação de associadas a uma Infra-estrutura de Chave Pública, podendo ser implementado através de diversos protocolos como: LDAP (Ligthweigth Directory Access Protocol), FTP (File Transfer Protocol) ou até mesmo HTTP (Hyper Text Transfer Protocol.

Hierarquia de Certificados

As Autoridades Certificadoras poderão ser interligadas formando uma estrutura hierárquica como visto na Figura 2.9. No modelo de hierarquia, as Autoridades certificadoras emitem certificados para as outras de um nível mais baixo. Em uma estrutura hierárquica todas as ACs conhecem a chave pública da AC que está na raiz da hierarquia.

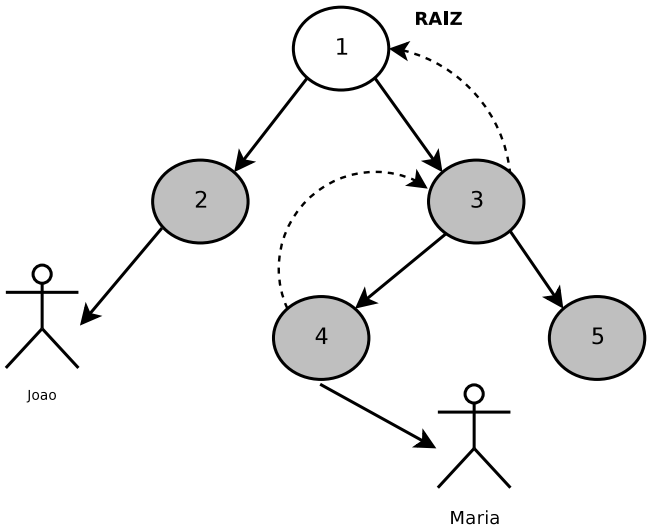

Figura 2.9. Hierarquia de Autoridade Certificadora.

A estrutura hierarquia permite que os certificados possam ser validados através do

caminho dos certificados da CA raiz (número 1 na figura). Vamos supor, de acordo com a figura, que João quisesse verificar o certificado de Maria. Para fazê-lo ele teria que validar o certificado de chave pública de Maria. Depois dessa verificação João teria que validar o certificado da Autoridade Certificadora de Maria, 3 na figura. Até que alcance a raiz que possui chave pública conhecida. A linha tracejada apresenta na figura o que chamamos de caminho de certificação.

Uma outra opção de interligação de Autoridades é a estrutura mista. Neste tipo de interligação várias Autoridades Certificadores se autenticam mutuamente, criando várias relações de confiança. Neste tipo de estrutura não existe a hierarquia nas relações de confiança entre as autoridades certificadoras. O principio de funcionamento é análogo ao anterior.

X.509 O X.509 faz parte de um conjunto de recomendações da IEFT para certificados de chave pública. A recomendação X.509 vem ao longo dos anos sofrendo uma modificações encontrando-se atualmente na sua versão 3. Um certificado X.509 é composto de informações assinadas pelo seu emissor sobre um determinado sujeito que são armazenadas em campos obrigatórios e opcionais. Para validar a autenticidade deste certificado verificamos a assinatura do emissor e, caso a assinatura seja considera verdadeira, as informações contidas no certificados serão consideradas confiáveis.

A Figura 2.10 representa um certificado de identidade X.509 na sua versão 3. Ele é composto de campos opcionais e obrigatórios como descritos a seguir.

Versão

Indica a versão do X.509 (atualmente 3). **Número de Série**

O número de série é um identificador único atribuído pelo emissor.

Algoritmo e Parâmetros

Contém o identificador do Algoritmo usado para assinar o certificado e seus parâmetros

Emissor

O Emissor é o nome único usado para identificar o certificado x.509.

Não antes de e Não depois de

Indicam o período de validade do presente certificado.

Sujeito

No Sujeito contém o identificador do Sujeito objeto desse certificado que possui a chave pública representada neste certificado.

Algoritmos, Parâmetros e Chave pública do Sujeito

Estes campos contém as informações relativas a chave pública do sujeito. Estes campos indicam o algoritmo e os parametros utilizados para gerar a chave pública.

Identificadores únicos do Emissor e do Sujeito

Esses campos são usados caso seja necessário reutilizar os nomes dos nomes do

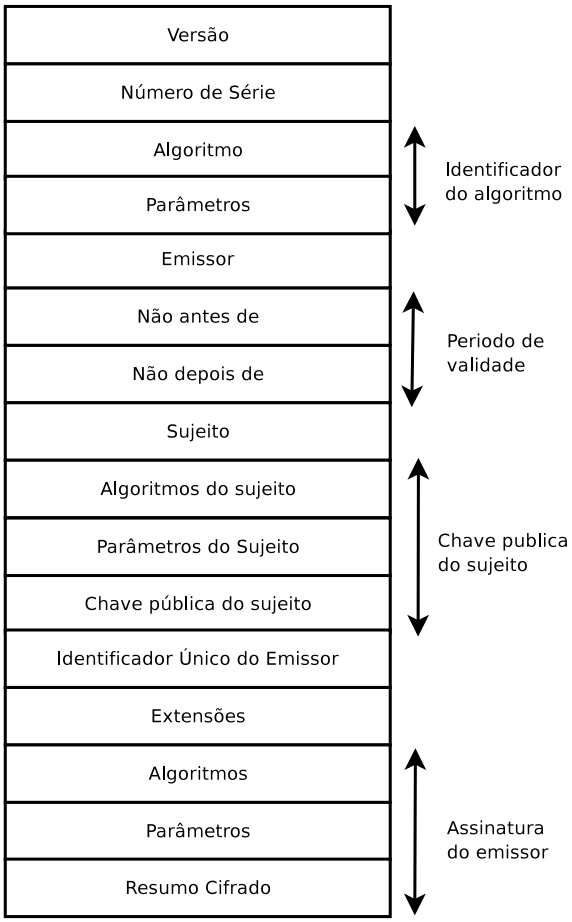

Figura 2.10. Campos que representam um Certificado de identidade X.509.

Sujeito e do Emissor.

Extensões

Esse campo é utilizado para permitir extensões do certificado. Através desse campo é permitido ao emissor implementar funcionalidades adicionais no certificado de acordo com as suas necessidades. Este campo poderia, por exemplo, ser usado para fornecer informações de autorização ao Sujeito do certificado.

Algoritmos, Parâmetros e Resumo Cifrado

Os campos Algoritmos, Parâmetros e Resumo Cifrado são usados para indicar quais algoritmos e parâmetros foram usados para assinar este certificado.

Como foi descrito anteriormente, o campo de extensão pode ser usado para permitir que informações de autorização a um certificado de identidade. Existe pelo menos dois bons motivos para isto não ser feito desta forma. O primeiro motivo é que o certificado de identidade possui um tempo de vida maior que o de autorização. Uma autorização pode deixar de existir e nem por isso um sujeito de um certificado de identidade deveria ter o seu certificado de identificação revogado.

Além do certificado de identidade, a recomendação X.509 permite a definição de

certificados de autorização. Os certificados de autorização contém atributos que especificam privilégios de um grupo, usuário, papel (role) ou outra informação de autorização associada ao proprietário desse certificado [Xavier, 2004].

2.3.2.4. Redes de Confiança (SPKI/SDSI)

O desenvolvimento do SDSI/SPKI foi motivado pela alta complexidade das infraestruturas de chave públicas, em especial o X.509. O SDSI foi projetado no MIT por Ronald Rivest e Butler Lampson [Rivest and Lampson, 1996]. O SDSI é uma infraestrutura de chaves públicas com espaço de nomes locais, o que dá a característica de descentralização. O SPKI foi desenvolvido por Carl Ellison e outros [Ellison et al., 1999] e é um sistema de autorização flexível e simples. Os dois projetos se uniram e formaram o SPKI/SDSI um sistema de autenticação e autorização que combina o espaço de nomes locais do SPKI e o sistema de autorização do SPKI.

O SPKI/SDSI usa como forma de representação as *S-expressions*. Uma s-expression é uma convenção para representar estruturas de dados ou expressões no formato de texto, como usado nas estruturas LISP. As *S-expressions* encapsulam elementos com parênteses cujos elementos podem ser cadeias de caracteres ou outra *S-expression*.

Uma chave pública representada sobre a forma de *S-expression* pode visto na Figura 2.5 a seguir. A primeira linha é o identifica que a *S-expression* é de uma chave pública. A segunda linha representa os algorimos de criptografia usados dentro dessa chave pública: algoritmo de chave pública RSA, padrão de criptografia PKCS1 e algoritmo de *hashing* MD5. Finalmente, o valor (e #11#) indica o valor do expoente da chave RSA. Os valores "n| ANeWQ0..."representa a chave criptográfica.

```
(public key
        (rsa pkcs1 md5 (e #011#)
                (n |ANeWQ0+7nhwMzuahgLPPMbOi6jUP0RPgZTzpLAhJ6qm/1DT1LVRYF78izo5z
                JqtTCB/yYoSExEEM2e8Anx7trz+wK6U4HdBcEwexjkXXo3BM9D433bpVm8iM61y
                8FMaYH743gvectGZ3BBBGNzH6KHAERXjW0te2y9UpT1GzWart|FMaYH743gvect
                GZAA|)
        )
)
```

Figura 2.11. Chave Pública SPKI/SDSI.

Nomes no SPKI/SDSI

No SDSI/SPKI a identificação é feita através de chaves públicas e não por um nome. O argumento para tal consideração é que o nome não é um identificador único (o caso dos homônimos) e em alguns casos podem também mudar. É bem comum as pessoas mudarem seus nomes, seja porque não gostam ou porque contraem matrimônio. Uma

chave pública, por outro lado, não pode ser considerada única, porem é pouco provável a coincidência da escolha aleatória de uma chave pública. Um tamanho de chave pública comumente usado possui cerca de 309 caracteres (1024 bits).

O SDSI associa uma chave pública a um nome no espaço de nomes local do usuário. Um nome pode estar associado a zero ou mais identificadores e somente serão conhecidos localmente, ou seja, não necessitam ser globalmente único. Um nome básico no SDSI é uma *S-expresssion* com dois elementos: a palavra *name* e o nome escolhido para identificá-lo. Por exemplo joão poderia identificar maria dentro do seu espaço de nome da seguinte forma: joão: (*name* maria).

O nome tem significado somente para quem o definiu, porém para ser identificado globalmente deve ser associado a uma chave pública. Assim se a chave pública definida na Figura 2.11 fosse usada para identificar maria, a *S-expression* que representa o nome completo SDSI visto na Figura 2.12.

```
(name
(public key
        (rsa pkcs1 md5 (e #011#)
                (n |ANeWQ0+7nhwMzuahgLPPMbOi6jUP0RPgZTzpLAhJ6qm/1DT1LVRYF78izo5z
                JqtTCB/yYoSExEEM2e8Anx7trz+wK6U4HdBcEwexjkXXo3BM9D433bpVm8iM61y
                8FMaYH743gvectGZ3BBBGNzH6KHAERXjW0te2y9UpT1GzWart|FMaYH743gvect
                GZAA|)
        )
)
maria)
```

Figura 2.12. Nome completo SPKI/SDSI.

Certificados SPKI/SDSI

Como uma solução totalmente distribuída o SDSI/SPKI permite uma grande flexibilidade nas definições de certificados e delegação. Primeiramente, o SPKI/SDSI é sistema igualitário, ou seja, não existe hierarquia entres os participantes. Cada usuário é responsável por gerenciar seus próprios certificados, ou seja, é uma autoridade certificadora. Assim eles possuem liberdade para gerar certificados e delegar acessos aos seus recursos.

Existem dois tipos de certificados no SPKI/SDSI: o Certificado de Nome (*Name Certs*) e o Certificado de Autorização. *Auth Certs*. O Certificado de Nome providencia autenticidade de um nome local, ou seja, ele certifica que o nome criado dentro do espaço de nomes do emissor é válido. O certificado de autorização concede uma autorização de acesso a um recurso pertencente ao emissor ao sujeito do certificado.

Um certificado de nome é composto por quatro campos: *issuer*, *identifier*, *subject* e *validity specification* [Clarke et al., 1999]. O *issuer* é a chave pública que assina o certificado. O *identifier* identifica o nome local que se está definindo. O *subject* é representado por uma chave pública ou um nome (chave seguida de um ou mais iden-

tificadores). Caso o *subject* não seja iniciado por uma chave pública é considerado que o nome pode ser encontrado dentro do espaço de nomes local. O *validity specification* corresponde ao período de validade do certificado é considerado válido; o *validity specification* também pode ser usado para verificação em lista de revogação de certificado. O Certificado de Nome em *S-Expression* tem a seguinte forma: (cert (issuer (name K N)) (subject S) <valid>). Onde: K é o *issuer*, N é o *identifier*, S é o *Subject* e <valid> é o *validity specification*. O exemplo a seguir cria um certificado de nome para maria no espaço de nome de joão.

O SPKI permite a definição de grupos. Cada grupo possui um nome e um conjunto de membros. Um grupo pode referenciar também outros grupos. Para criar um grupo, o dono do grupo emite, para cada membro do grupo, um certificado definindo o nome local do grupo. A possibilidade de criação de grupo facilita o controle de acesso aos recursos, com esta estrutura pode-se autorizar (ou desautorizar) todas as pessoas do grupo apenas fazendo isto para o grupo.

Um Certificado de Nome consiste de cinco campos: *issuer*, *subject*, *delegation* , *tag* e *validity specification*. Os dois primeiros tem função análoga ao Certificado de Nome explanado anteriormente, sendo que o Subject pode também indicar um grupo.. O campo *delegation* indica se o certificado pode ser delegado ou não. O *tag* especifica que tipo de autorização (ou autorizações) o sujeito do certificado receberá. O *validity specification* tem função análoga ao Certificado de Nome.

Através da indicação do campo *delegation* um Certificado de Autorização quando criado pode permitir que o sujeito do certificado possa delegar seus direitos. A Figura 2.13 mostra um exemplo de delegação que poderia ocorrer num ambiente com SPKI/SDSI. A impressora I (ou um sistema de gerenciamento que a represente) emite um certificado com delegação para o gerente G permitindo o direito de imprimir - I(I,D)G. O gerente, por sua vez, delega este direito a um funcionário F que confia, gerando um novo certificado para o funcionário - G(I,D) e entrega o certificado que gerou junto com o certificado recebido da impressora. Um estagiário não muito confiável solicita o direito de imprimir para a funcionária, esta o faz mas com a ressalva de não propagar o direito de delegação. O estagiário para imprimir entrega toda cadeia de confiança que foi gerada no processo e sua assinatura. A impressora confirma todas as assinaturas e garante o direito de impressão para o estagiário.

A próxima subseção apresentará estudos de caso de implementações de grades computacionais representando o estado da arte na área.

2.4. Estudo de casos

Nesta seção apresentamos estudos de caso de algumas grades computacionais existentes. Uma breve descrição de cada arquitetura é mostrada. A seguir, decrevemos a implementação de segurança de cada uma delas.

2.4.1. Globus

O Globus [Foster and Kesselman, 1997, Globus, 2004] é atualmente o projeto de maior impacto na área de Computação em Grade. O Projeto Globus envolve muitas ins-

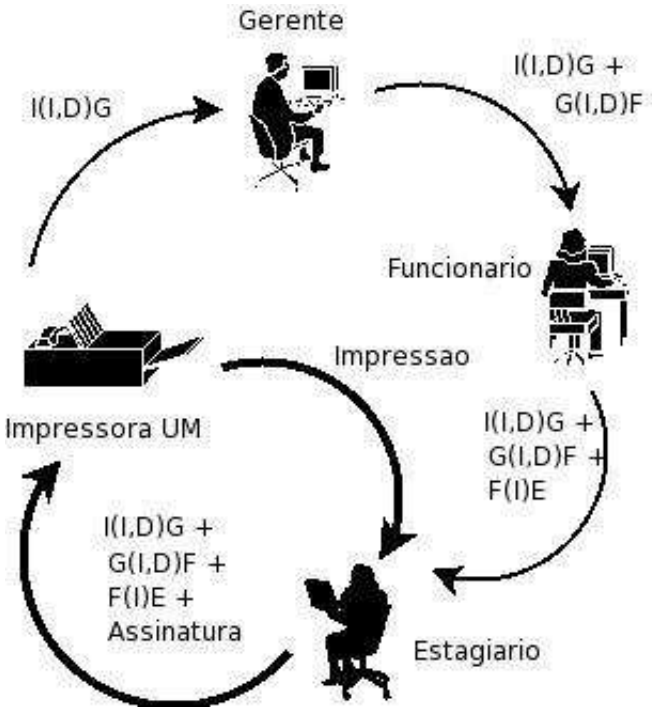

Figura 2.13. Delegação de um Certificado.

tituições de pesquisa da Europa, da Ásia e pricipalamente dos Estados Unidos, dentre elas podemos citar: Laboratório Nacional de Argonne (ANL), Universidade de Chicago, Universidade do Sul da Califórnia (USC) e o Laboratório de Computação de Alto Desempenho da Universidade do Nordeste de Illinois e o Imperial College na Inglaterra. Grandes intituições da indústria também apoiam o projeto tal como: IBM, Microsoft e Cisco.

O projeto Globus inclui o desenvolvimento de um sistema de Computação em Grade denominado Globus Toolkit. O *Globus Toolkit* (GT), é um pacote que permite a construção de sistemas de grade de modo incremental. O termo *toolkit* é utilizado pelo fato dele permitir a configuração e personalização de grades especializadas e aplicações. O Globus Toolkit está atualmente na versão 4.0, que apresentamos nesta subseção.

A versão atual do Globus Toolkit é uma implementação do OGSA (*Open Grid Services Architecture*), uma arquitetura padrão que visa a construção de uma infra-estrutura para grades baseada em serviços [Foster et al., 2002]. Serviços são entidades que disponibilizam suas funcionalidades a usuários e aplicações através da rede. No modelo OGSA, qualquer objeto é representado por um serviço, tais como: dispositivos de armazenamento, redes, programas, entre outras entidades.

A base da definição de serviços para o OGSA é denominada *Grid Service*. O *Grid Service* é uma extensão de um *Web Service* que mantém informações do estado dos serviços. A definição de estados permite a distinção entre a instância de um serviço de outras que providenciam uma mesma interface.

As interfaces básicas e o comportamento de um *Grid Service* é definido através

do WSRF (*Web Service Resource Framework*) [Foster and Czajkowski, 2005]. O WSRF é uma evolução do OGSI (*Open Grid Service Infrasctructure*) que é um padrão utilizado nas versões anteriores do GT. O WSRF separa as funcionalidades do OGSI em diversas especificações e inclui algumas novas especificações que surgiram. O WSRF define uma família de especificações para acessar recursos usando serviços Web.

A seguir apresentamos o GSI, uma implementação de arquitetura de segurança baseada no OGSA.

2.4.1.1. GSI - Grid Security Infrastructure

O Globus disponibiliza um serviço de segurança denominado GSI (*Globus Security Infrastructure*) [Foster et al., 1998], uma implementação da arquitetura de segurança baseada no OGSA. Assim como outros serviços do Globus, o GSI é uma especificação abstrata que pode ser implementada sobre diferentes mecanismos de segurança.

A Figura 2.14 mostra o resumo da arquitetura do GSI [Ian, 2005]. O GSI utiliza diversos padrões para implementar funcionalidades como autenticação, delegação proteção de mensagens e autorização.

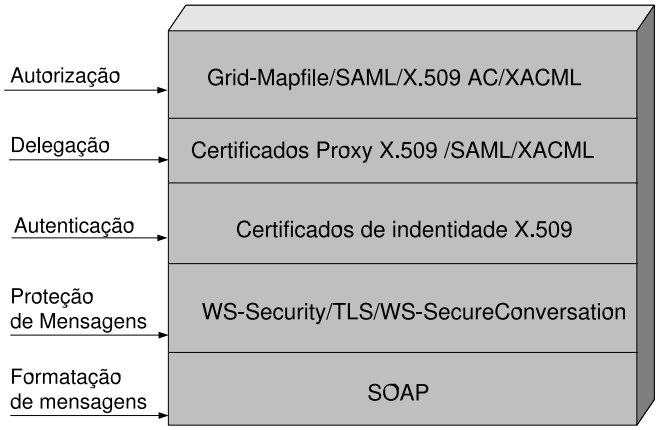

Figura 2.14. Arquitetura do GSI.

A autenticação do GSI é feita através de credenciais X.509. O X.509 é utilizado no GSI para permitir o estabelecimento de confiança entre domínios de segurança diferentes. Além da autenticação X.509, presente também nas versões anteriores do Globus Toolkit, a versão 4 permite a autenticação de usuários utilizando identificador de usuário e senha. Porém quando este método é usado perde-se a possibilidade de usar mecanismos de segurança que incluem confidenciabilidade e integridade de dados.

Para permitir a delegação dinâmica o GSI estende o conceito de *proxy* do X.509 que permite ao usuário atribuir uma nova identidade X.509 ao usuário e então delegar alguns de seus atributos de segurança a esta nova identidade [Welch et al., 2004]. Este mecanismo permite a criação de novas identidades e credenciais sem a intervenção do administrador da rede. A vantagem de se estender o padrão X.509 é a possibilidade de

utilização de bibliotecas já existentes com uma modificação bem pequena.

A tabela 2.1 resume as diferenças entre o certificado X.509 e o certificado x.509 com proxy[Welch et al., 2004]. No campo *Issuer* o Certificado Proxy é identificado pelo chave pública do certificado ou por outro Certificado X.509 Proxy. Esta opção permite que o certificado seja criado dinamicamente por possuidor de certificado, sem a intervenção das CAs. Ao contrário do padrão X.509, onde o sujeito de um certificado é definido pela CA, o campo *Subject Name* do certificado permite a definição de sujeitos dentro do escopo de nomes do emissor do certificado. A restrição do escopo deve ser feita para que os certificados sejam únicos. A chave pública do Certificado X.509 Proxy é distinta da chave pública do seu emissor.

Tabela 2.1. Comparação entre X.509 padrão e X.509 proxy estendido pelo Globus Toolkit.

Atributo do Certificado	**X.509**	**X.509 Proxy**
Issuer/Signer	Autoridade Certificadora	Certificado de chave pública ou Certificado de outro Proxy
Subject Name	Definido pela Autoridade certificadora	Definido no espaço de nomes do dono do certificado
Delegação do Emissor	——	Define as condições dos direitos delegados
Key pairs	Par de Chave única	Par de chave única

O Certificado de Proxy acrescenta duas características adicionais ao Certificado X.509 padrão: autenticação única e delegação. A autenticação única dispensa a digitação freqüente de uma senha para provar a autenticidade de um usuário. A delegação, como já definido anteriormente, permite que o servidor se conecte a qualquer recurso em nome do cliente.

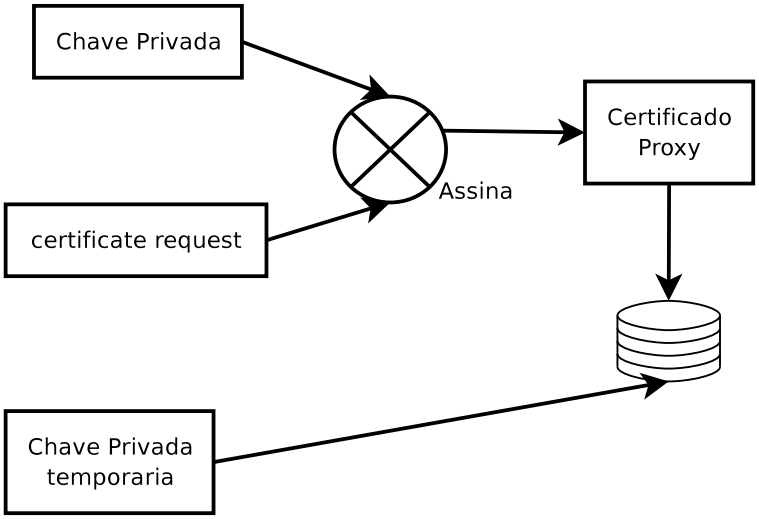

Figura 2.15. Protocolo de Autenticação Única.

A autenticação única permite o conceito de autenticação na Grade para reduzir o número de vezes em que o usuário precisa acessar sua chave privada [4]. Com o Certificado

[4]O acesso a chave privada é uma ação de risco por isso ela é geralmente protegida por senha, em um ambiente de grade isso pode não ser viável.

Proxy, o usuário autentica uma vez para criar o Certificado Proxy (que possui um par de chaves diferente do seu criador) e o usa inúmeras vezes sem precisa usar a chave privada do seu criador. Para aumentar a segurança o Certificado Proxy possui um tempo de vida curto, e o compromentimento de sua chave não chega ser um problema grave de segurança.

A autenticação única no Globus Toolkit é mostrada na Figura 2.15. Inicialmente um novo par de chave pública e privada é criado e associado a um *certificate request*[5]. A seguir, o usuário usa sua chave privada para assinar o Certificado Proxy recém criado. Finalmente, o Certificado Proxy é armazenado de forma protegida em disco.

A delegação de direitos no Globus Toolkit também pode ser feita sobre um canal de comunicação [Welch et al., 2004] como pode ser visto na Figura 2.16. Inicialmente, os dois envolvidos na delegação negociam um canal seguro de comunicação. O delegado, aquele que recebe a delegação, cria um par de chaves e um *certificate request*. O delegado então envia o *request certificate* sobre o canal seguro. Ao receber o *request certificate*, o delegador define as restrições nos direitos doados e usa a chave privada associada ao seu Certificado Proxy para gerar o novo Certificado Proxy. O delegador envia o Certificado sobre o canal seguro de rede. Ao receber o certificado, o delegado coloca em um lugar seguro, junto a chave privada criada no inicio do processo.

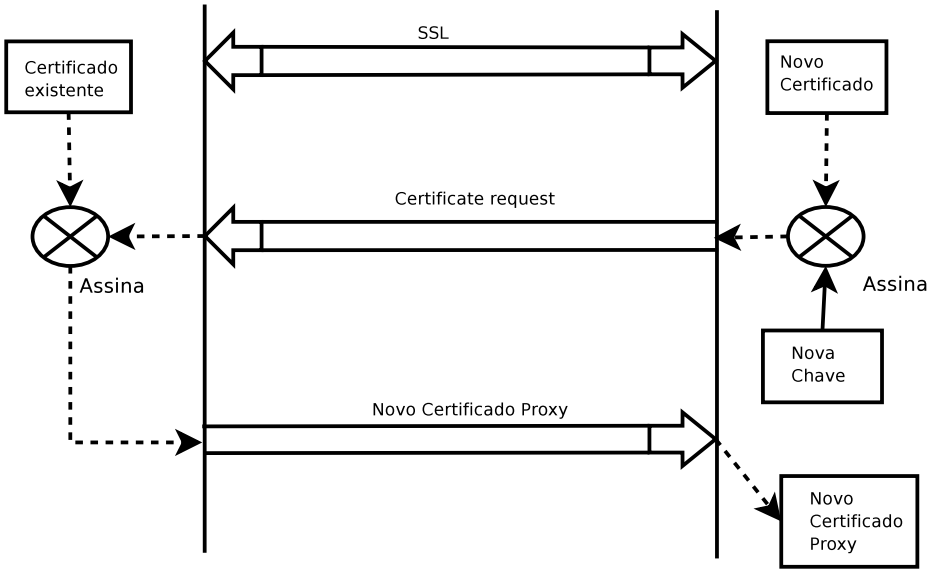

Figura 2.16. Delegação de Certificado Proxy

Para permitir a comunicação entre as federações, o GSI usa *gateways* que fazem tradução entre credenciais X.509 e outros mecanismos. O KCA (ou KX.509) (`http://www.citi.umich.edu/projects/kerb_pki`), desenvolvido na **University of Michigan** é um sistema baseado no Kerberos que permite que usuários adiquiram certificados X.509 usando *tickets* Kerberos. O usuário se autentica normalmente no Kerberos e solicita uma credencial para acessar o serviço do KCA. O KCA recebe o *ticket*, determina

[5]Especificação padrão para codificar requisições de certificados, incluindo o nome da pessoa que requisita o certificado e sua chave pública.

a identidade do usuário, gera um certificado de tempo curto (*short-lived* X.509) e envia de volta ao usuário que solicitou.

O GSI implementa troca segura de mensagens na rede de duas formas: no níveis do transporte e da mensagem. A comunicação no nível do transporte usa o TLS (Transport Layer Security)[Dierks and Allen, 1999] para a criação de comunicação segura fim a fim. A comunicação no nível da mensagem usa padrões de segurança baseado em *Web Services* [Booth et al., 2004] e SOAP (Simple Object Access Protocol) [W3C, 2000].

O TLS é baseado no protocolo *Secure Socket Layer* (SSL) desenvolvido pela empresa Netscape Corporation. O TLS é um protolo de transporte seguro que provê privacidade e integridade e é usado para encapsular mensagens em protocolos de alto nível. O TLS permite que as partes envolvidas realizem autenticação mútua e negociação de algoritmos de criptografia, criando um canal seguro de comunicação. O TLS foi projetado de forma a ser um protocolo eficiente quanto ao uso de CPU e o uso da rede, o que faz com que ele seja uma boa opção de uso em grades computacionais.

O *Globus Toolkit* implementa as especificações *WS-Security* e *WS-SecureConversation* para permitir comunicação segura no nível de mensagem. O padrão *WS-Security* é um arcabouço que permite a criptografia e assinatura de mensagens sobre SOAP. Ele especifica um mecanismo que associa um *token* de segurança às mensagens. Para o padrão *WS-Security*, o token é uma estrutura de dados extensível, codificado em binário, usado para representar de forma transparente os vários mecanismos de segurança. A integridade das mensagens é garantida através de *XML Signature* e a confidenciabilidade usando *XML Encription* [Foster and Kesselman, 2003]. O *WS-SecureConversation* gerencia a criação de contextos de segurança e cria chaves que podem ser usadas neste contexto. Este contexto pode então ser usado para proteger mensagens subseqüentes sem que o processo de autenticação seja repetido a cada momento.

A autorização no *Globus Toolkit* é feita através do uso de SAML (*Security Assertions Markup Language*) e de Grid mapfiles. O SAML define a sintaxe e semântica de afirmações feitas sobre um sujeito por uma entidade. O GSI usa essa linguagem para permitir a comunicação com autoridades certificadoras e receber as informações a respeito das autorizações que um cliente tem sobre um determinado serviço. O uso do Grid-mapfile é mantido por compatibilidade a versão 2 do GT e associa um determinado usuário a nomes qualificados que aparecem em certificados X.509. Por exemplo, a entrada "/O=Grid /OU=Ime /OU=simpleCA-ime.usp.br /OU=localdomain /CN=Jose Braga" jrbraga associaria o usuário jrbraga ao correspondente identificador X.509.

O Globus Toolkit introduz o servidor CAS (*Comunity Authorization Server*) que é responsável pelo gerenciamento de políticas de acesso para os recursos da comunidade [Pearlman et al., 2002]. Dependendo da política da comunidade, o CAS pode ser controlado por um ou mais administradores. O CAS permite que as tarefas de administração possam ser distribuídas. Um administrador pode, por exemplo, ser responsável pelo registro de usuários e recursos em um servidor CAS, mas não ter o direito de associá-los a grupos que possuem direitos específicos, podendo essa autorização ser feita por um outro administrador.

A Figura 2.17 representa um usuário solicitando ao servidor CAS uma requisição

de capacidade (*capability*). Essa capacidade permite a ele executar um conjunto de ações específicas. O servidor CAS, caso a política de acesso definida para esse usuário permita, devolve ao usuário a capacidade solicitada. De posse dessa autorização, o usuário pode usar a capacidade de acordo com os direitos permitidos ou delegá-la a um terceiro.

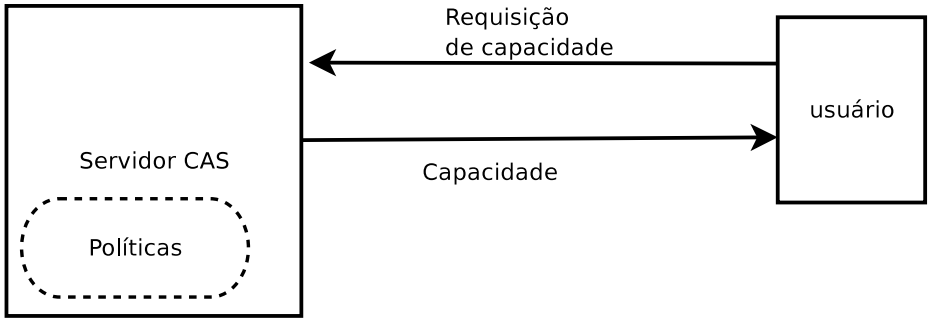

Figura 2.17. Usuário faz uma requisição de capacidade ao servidor CAS.

A Figura 2.18 mostra um usuário submetendo uma capacidade a um provedor do recurso. O usuário faz uma requisição ao dono do recurso provando que possui direitos de acesso. A seguir, o provedor verifica se a capacidade apresentada pelo usuário e se as políticas definidas localmente permitem o uso do recurso. Caso isso seja possível, o usuário finalmente recebe o direito de utilizar o recurso.

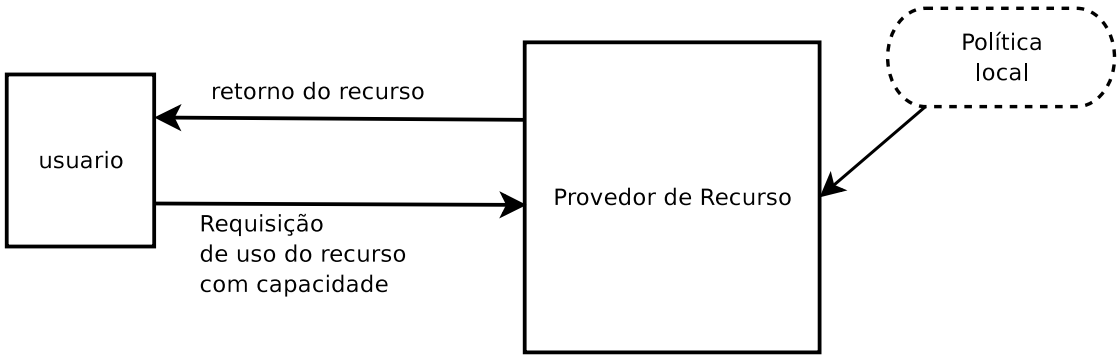

Figura 2.18. Cliente requisitando uso de recurso.

2.4.2. Legion/Avaki

O Projeto Legion [Lewis and Grimshaw, 1996] foi desenvolvido na Universidade da Virginia com o objetivo de prover abstrações que permitissem uma visão única para milhares de computadores interligados em um sistema distribuído. Desde o início do projeto, o Legion foi criado com o intuito de permitir a transferência de sua tecnologia para a indústria; isto ocorreu em 2001 através da empresa AVAKI fundada pelos seus idealizadores. A companhia detém os direitos sobre o Legion que passou a ser chamado de Avaki.

No Legion, todas as entidades da Grade são representadas por objetos (no sentido de Orientação a Objetos), tais como: computadores, usuários, equipamentos, dispositivos de armazenamento e aplicações. Estes objetos comunicam-se entre si através de chamadas

a métodos assíncronos. Cada método possui uma assinatura própria que descreve os seus parâmetros e valores devolvidos. Um objeto é descrito por um conjunto de interfaces que define suas classes. O Legion utiliza uma IDL (*Interface Definition Language*) para descrever seus objetos e é um sistema reflexivo onde as próprias classes são objetos.

Os objetos Legion são persistentes e podem estar associados a dois estados: ativo ou inativo. Os objetos ativos ficam presentes na memória, possuem um processo associado e podem responder a chamadas a seus métodos. Os objeto inativos são armazenados em algum dispositivo de armazenamento através de sua representação seriada (OPR - *Object Persistent Representation*). Cada OPR é endereçado por um endereço de representação persistente (OPA - *Object Persistent Address*).

Cada objeto Legion possui um identificador LOID (*Logical Object Identifier*) e um endereço de objeto (LOA - *Legion Object Address*). O LOID é um endereço único, atribuído pela classe do objeto no momento de sua criação. O LOA provê um ou mais protocolos de rede utilizados para a comunicação concreta com um objeto. Por exemplo, o LOA poderia utilizar o protocolo TCP indicando o endereço IP e a porta onde o objeto está localizado.

O Legion define as interfaces e funcionalidade para um conjunto de classes bases denominadas objetos do núcleo. Essas classes básicas oferecem serviços básicos para os objetos que as herdam. Os objetos do núcleo são:

- *Legion Object e Legion Class* - definem um conjunto de métodos que todos os objetos e classes devem implementar. O *Legion Object* define o método $may_I()$ usado para controle de segurança das chamadas de execução.
- *Legion Hosts* - Quando um objeto *Legion Host* é ativado ele cria um processo para contê-lo. Cada recurso computacional pode estar associado a mais de um objeto *Host*. O *Legion Hosts* tem como responsabilidade criar e gerenciar processos para objetos Legion ativos.
- *Legion Vaults* - representam o armazenamento persistente dos demais objetos Legion. Quando um objeto é seriado, sua representação (OPR) é armazenada em disco por objetos *Legion Vaults*.
- *Implementation Objects* - encapsulam aplicações a serem executadas no Legion. O Legion utiliza metadados para apresentar as características importantes de uma aplicação, tais como: formato da aplicação (por exemplo, binário ou byte-code), plataforma e requisitos de hardware.

2.4.2.1. Segurança

O identificador de objetos LOID possui um campo que armazena uma chave pública, que vai ser a base para a autenticação e da autorização do Legion. Através do LOID é possível se comunicar de forma segura com um objeto obtendo a chave criptográfica armazenada. Como a chave pública no Legion é associada ao nome do objeto, um atacante não pode subistituir uma nova chave em um identificador de objeto conhecido, por que

se uma parte do LOID é alterada, incluindo a chave, um novo LOID é criado e não será conhecido.

O Legion usa o paradigma de usuário e senha para permitir a autenticação do usuário. Para se autenticar, o usuário fornece um identificador de conta (*login*) e uma senha. O nome do usuário é associado ao que o Legion chama de objeto de autenticação que vai agir como o procurador do usuário na Grade. O objeto de autenticação possui a chave privada, a senha criptografada e o perfil do usuário no seu estado persistente. A senha oferecida pelo usuário é comparada à senha armazenada no estado persistente do procurador para negar ou permitir o acesso à Grade. O estado de autenticação do objeto é armazenado de forma persistente no disco na máquina.

Uma outra forma de autenticação no Legion é através do LDA (*Legion Data Access*). O LDA cria um mapeamento entre o identificador do usuário no Unix e o LOID de um objeto de autenticação para acessar a dados da grade usando o protocolo NFS. Quando um cliente NFS monta um DAP ele mapeia o espaço de nomes no sistema de arquivo local do seu host, providenciando um acesso aos dados da Grade de forma transparente. O DAP fornece suporte ao mecanismos de segurança do Legion, o controle de acesso é feito através de credenciais assinadas e as interações com a Grade pode ser assinada.

O Legion usa credenciais para permitir a delegação de direitos. O Legion define uma forma de transferência de direitos através de credenciais para objetos. O credencial é uma lista de direitos assinada pelo usuário ou pelo seu procurador. O recurso verifica a credencial e segue a cadeia de confiança até conferir a assinatura, decidindo se garantirá o direito ao recurso.

A autorização do usuário no Legion é feita através de lista de controle de acesso (LCA). Para cada função ou ação associada a um objeto existe uma lista de permitidos e negados (*allow* e *deny*). Cada lista contém o nome de outros objetos ou um diretório que contém uma lista de objetos. Caso um diretório contenha objetos de autenticação (um objeto de autenticação é o representante do usuário na Grade) então ele atua como grupo de usuários. Se um objeto estiver presente simultaneamente nas duas listas, o objeto será negado de efetivar a ação.

O protocolo SSL é usado para permitir a comunicação segura no Legion. Os dados são protegidos no Legion de três formas: privada, protegida e nenhuma. Na primeira, mais restritiva, todas as mensagens são completamente criptografadas, garantindo confidencialidade e autenticidade. No modo protegido, as mensagens são protegidas de modificação através de funções de resumo (*hashing*). E finalmente, o Legion permite que as mensagens possam ser transmitidas sem criptografia, com exceção das credenciais.

O Legion providencia um mecanismo de *sandboxing* para a proteção das máquinas hospedeiras. O *sandbox* faz um isolamento do identificador real do usuário e do sistema de Grade na maquina hospederia. O *sandboxing* do Legion permite a definição de identificadores genéricos, portanto que não estão associados a nenhuma conta na máquina.

2.4.3. Condor

O sistema Condor[6] foi criado no início dos anos 80 na Universidade de Wisconsin como um sistema para computação distribuída caracterizado por deixar o controle das máquinas sob a responsabilidade dos seus proprietários. O Condor (Condor high-througput computing system) é um sistema de computação batch distribuído para computação intensiva, com mecanismos para gerenciamento de tarefas, políticas de escalonamento, gerenciamento e monitoramento de recursos, etc.

A disponibilidade de grande montante de recursos tolerante a falhas por prolongados períodos de tempo (high throughput) associada a computação oportunista (aproveitamento de recursos ociosos) são características fundamentais do Condor. Essas características são implementadas através dos mecanismos de *ClassAds*, migração e *checkpoint* de tarefas e chamadas de sistema remotas.

A linguagem ClassAd provê meios eficientes para comparar os recursos oferecidos com as solicitações de recursos recebidas, ou seja. na submissão de tarefas. O Checkpointing periódico de tarefas provê tolerância à falhas e permite migração de tarefas entre máquinas para escalonamento preemptivo de baixa custo [Krueger, 1988]. Condor implementa chamadas de sistema remotas, um mecanismo de sandbox móvel, para redirecionar, quando executando tarefas em máquinas remotas, chamadas de callback relacionadas a tarefas de I/O para a máquina que submeteu a tarefa. Com esse mecanismo os usuários não precisam disponibilizar arquivos de dados em estações de trabalho remotas antes da execução do Condor nessas máquinas.

Condor-G [Frey et al., 2002] é um encontro das tecnologias Condor e Globus projects. Protocolos para comunicação inter-domínio e acesso padronizado a diversos sistemas batch remotos são contribuições do Globus. O Condor-G incorpora do Condor os conceitos de submissão e alocação de tarefas, recuperação de erros e criação de um ambiente de execução amigável. No relacionamento com o Globus Toolkit, o Condor-G pode ser usado como um serviço confiável para gerenciamento e submissão de tarefas para um ou mais sites enquanto o sistema Condor high-throughput pode ser usado como um serviço de gerenciamento da estrutura subjacente para um ou mais sites, com o Globus Toolkit como elo entre eles.

2.4.3.1. Segurança

O Condor possui uma biblioteca chamada CEDAR que permite a clientes e servidores negociar e usar diferentes protocolos de autenticação tais como Kerberos, GSI, chaves públicas, autenticação via redes ou nós confiáveis e sistema de arquivos. Essa biblioteca implementa um conjunto de interfaces de autenticação simples e extensível. A biblioteca CEDAR possui a capacidade de negociar integridade de dados e algoritmos de privacidade separado do protocolo de autenticação. A Figura 2.19 mostra a Arquitetura de Segurança do Condor.

A autenticação no Condor baseada em sistema de arquivos é implementada da seguinte forma. Um processo *daemon* utiliza a propriedade do arquivo para decidir a

[6] `http://www.cs.wisc.edu/condor/`

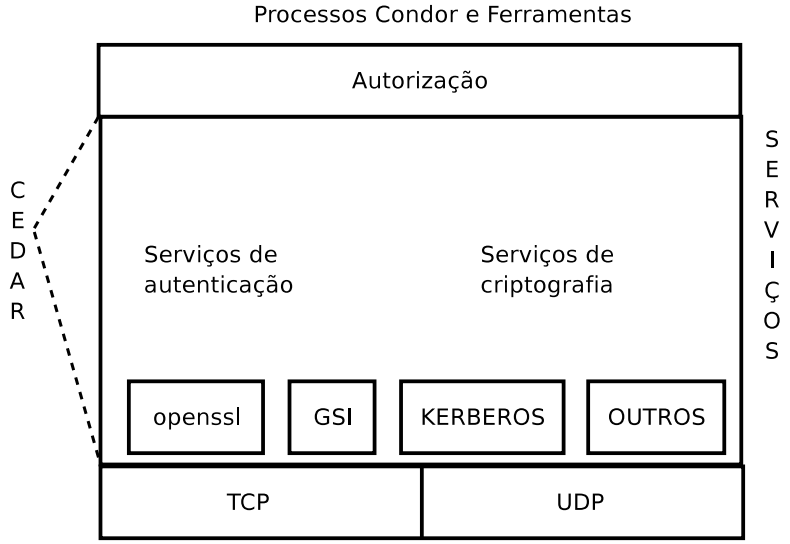

Figura 2.19. Arquitetura de Segurança do Condor.

identidade. Quando um usuário deseja entrar na grade, o processo solicita a escrita de um arquivo em um diretório de escrita temporária. Comparando o dono do arquivo e o nome que o usuário indicou, ele decide sobre a autorização.

A execução de uma tarefa no Condor é necessariamente cooperativa. Certas informações são conhecidas unicamente pela máquina de execução, tais como quais sistemas de arquivos, redes e bases de dados podem ser acessados, enquanto somente a máquina de submissão conhece em tempo de execução os recursos necessários para a tarefa. Essa cooperação conhecida como *split execution* é realizada no Condor pelos componentes shadow e sandbox.

Condor provê o mecanismo chamado *shadow*, para proteção dos usuários, no site de submissão. O shadow providencia todo o necessário para especificação da tarefa em tempo de execução: o executável, os argumentos, o ambiente, arquivos de entrada, etc. Nenhum deles conhecido fora do agente de execução até o momento real da execução.

O *sandbox* no site de execução deve proteger proprietários de máquinas, impedindo acesso indevido aos recursos. O sandbox é formado por dois componentes distintos: sand, responsável por criar o ambiente apropriado à execução da tarefa e o componente box, responsável pela proteção dos recursos de possíveis danos que uma tarefa maliciosa possa causar.

Em versões iniciais, Condor restringia a execução de tarefas a uma parte limitada do sistema de arquivos através do comando *chroot* do Unix. Atualmente, as tarefas são executadas sem restrições no sistema de arquivos, mas com restrições no login. Essa abordagem possibilita contudo, em uma máquina multi-cpu executando múltiplas tarefas simultaneamente, o seqüestro de uma tarefa de usuário por outro usuário malicioso, porque compartilham um mesmo ID de usuário, ainda que de uma conta nobody padrão com poucos privilégios. Condor evita esse problema alocando dinamicamente IDs de usuário para cada tarefa em execução. No Unix, isso exige intervençaõ do administrador,

enquanto em ambiente Windows, Condor aloca usuários em tempo de execução. Condor também usa o conceito de domínio de ID de usuário para um conjunto de máquinas compartilhando uma mesma base de dados de usuário.

2.4.4. InteGrade

O InteGrade [7] é um sistema de Grade motivado principalmente pela necessidade de aproveitamento de recursos computacionais compartilhados (computadores pessoais, estações de trabalho), em geral subutilizados – com predomínio de períodos ociosos – para execução de aplicações com grande demanda por tais recursos. As características do InteGrade incluem suporte para uma gama de aplicações paralelas e mecanismos que buscam minimizar a percepção, pelos proprietários dos recursos compartilhados, de qualquer possível perda de qualidade de serviço [Goldchleger, 2004].

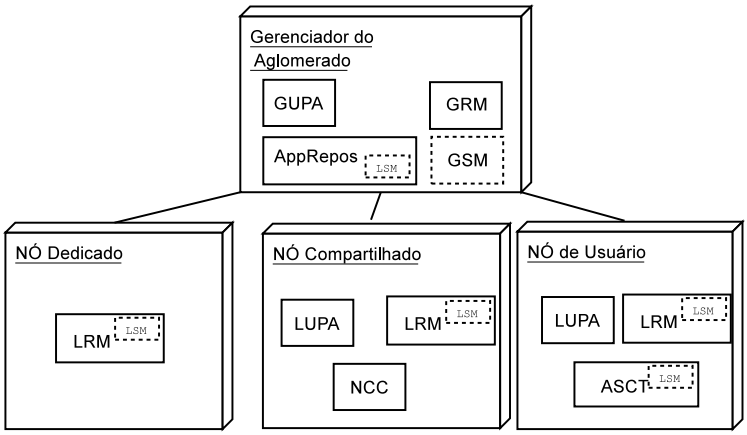

Figura 2.20. Arquitetura do InteGrade com os módulos de segurança.

Uma grade do InteGrade é constituída de aglomerados (*clusters*) de computadores organizados de forma hierárquica (Figura 2.20). Nesta seção apresentamos uma visão resumida da sua arquitetura através da descrição dos componentes de um aglomerado e dos módulos implantados nestes componentes.

- Gerenciador do aglomerado: responsável por gerenciar o aglomerado e pela comunicação com gerenciadores de outros aglomerados. Este nó pode ser distribuído para balanceamento de carga ou replicado para tolerancia a falhas.
- Nó dedicado: reservado para as aplicações da grade.
- Nó compartilhado (fornecedor de recursos): tipicamente uma estação de trabalho, disponibiliza parte de seus recursos para execução de aplicações dos usuários da grade.
- Nó de usuário: pertence ao usuário da grade que submete aplicações à grade.

O *Local Resource Manager* (LRM) e o *Global Resource Manager* (GRM) cooperam no gerenciamento dos recursos de um aglomerado. O LRM é executado nos nós do

[7] `http://gsd.ime.usp.br/inteGrade`

aglomerado, coletando informações sobre o estado do nó (uso de CPU, disco, memória, recursos de rede, etc) e as envia periodicamente ao GRM. Este último executa no nó gerenciador do aglomerado e usa as informações enviadas pelos LRMs para escalonamento das aplicações submetidas à grade.

LRM e GRM também colaboram no protocolo de alocação de recursos e execução de aplicações. O GRM seleciona nós candidatos para executar uma aplicação submetida pelo usuário da grade, com base nos requisitos da aplicação e na disponibilidade de recursos. Estas últimas informações são coletadas pelos LRMs nos nós fornecedores de recursos.

De modo semelhante à cooperação LRM/GRM, os módulos *Local Usage Pattern Analyzer* (LUPA) e *Global Usage Pattern Analyzer* (GUPA) também trabalham cooperativamente na análise de padrões de uso das máquinas para contribuir para um escalonamento eficiente na grade [Goldchleger, 2004].

O módulo *Application Repository* (*AppRepos*) permite o armazenamento e recuperação de aplicações.

2.4.4.1. Segurança

Os módulos que implementam os serviços de segurança são o LSM (*Local Security Manager*) e o GSM (*Global Security Manager*). Ambos utilizam a API GSS para obter os contextos de segurança entre o repositório de aplicações e os módulos que com ele interagem. O GSS, por sua vez, tem seus serviços implementados através do Kerberos. A implementação atual do repositório seguro utiliza a versão 5 do Kerberos desenvolvida pelo MIT[8] para a linguagem C. Para a linguagem Java, essa implementação utiliza a API GSS Java.

O GSM é responsável por iniciar e gerenciar os contextos. Cada cliente, através do LSM, possui um contexto de segurança com o GSM. Todas as trocas de mensagens entres os módulos são feitas utilizando esses contextos. O LSM pode efetuar quatro operações básicas: assinar e verificar uma mensagem enviada com o contexto que ela possui, ou ainda, assinar e verificar uma mensagem enviada sob um outro contexto. Mais especificamente, o repositório usa o GSM (via o LSM) para verificar e assinar os arquivos executáveis das aplicações dos seus clientes, enquanto estes clientes usam o LSM para assinar e verificar arquivos durante o armazenamento e recuperação do repositório.

O protocolo de armazenamento de uma aplicação do InteGrade é apresentado na Figura 2.21. O ASCT utiliza o LSM para assinar o arquivo e solicitar ao repositório de aplicações o seu armazenamento (1,2,3). O repositório de aplicações verifica a assinatura deste binário através do LSM (4), que o faz através do GSM (4.1). Uma vez o arquivo verificado com sucesso, o repositório de aplicações calcula um resumo do binário através de uma função *hash* conhecida[9], por exemplo, MD5, (5) e o armazena a seguir no sistema de arquivos (6). Ao final do processo de armazenamento em disco, o repositório de

[8] http://web.mit.edu/kerberos/www

[9] Para garantir que essa chave não possa ser gerada pelo atacante, o GSM adiciona ao arquivo uma chave que é obtida pela API GSS. Essa é uma técnica bastante utilizada em sistemas de segurança [Terada, 2000].

aplicações envia a identificação (ID) assinada da aplicação (7), que por sua vez também é verificada (8). Caso qualquer uma das operações falhar é gerada uma exceção que é devidamente tratada e registrada em arquivo de *log*.

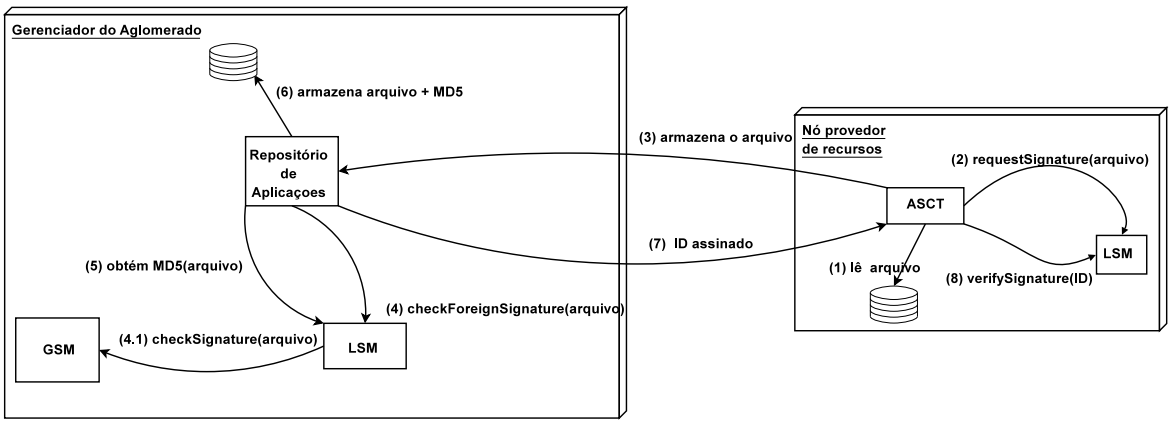

Figura 2.21. Protocolo de armazenamento de um executável de uma aplicação.

A Figura 2.22 apresenta o protocolo de recuperação de um arquivo. Uma vez que um determinado LRM recebeu a solicitação de execução de uma aplicação [Goldchleger, 2004], ele deverá requerer o binário compatível com a sua plataforma. De posse do ID da aplicação, o LRM o assina e indica ao repositório de aplicações o arquivo executável desejado (1,2). O Repositório obtém o arquivo desejado e verifica sua integridade através da função de *hash* (3,4). Antes de enviar o arquivo executável ao LRM, o repositório assina o binário através do LSM (5), que repassa essa função para o GSM (5.1), pois a assinatura deve ser feita através do contexto criado para o LRM que fez a requisição. Ao receber o arquivo, o LRM verifica sua assinatura (6,7) e o armazena em disco para execução (8).

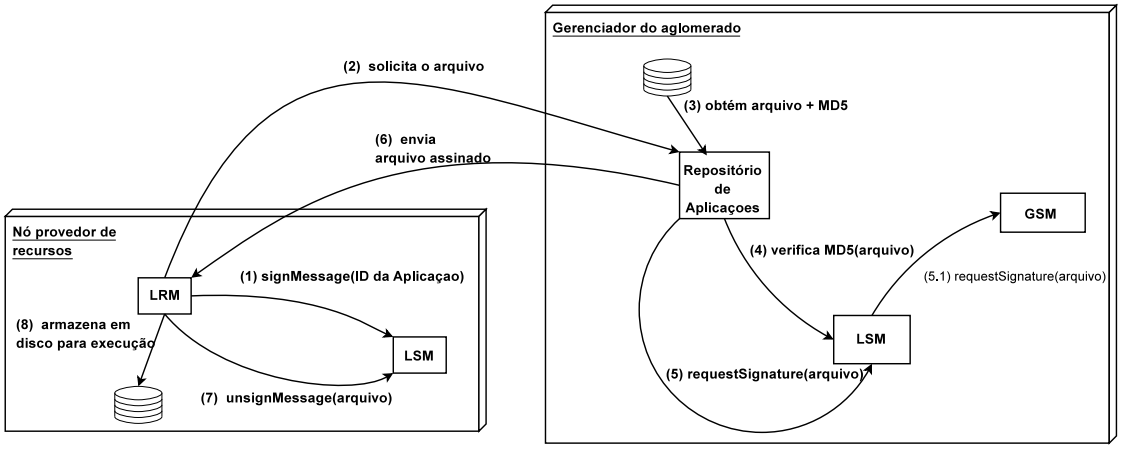

Figura 2.22. Protocolo de recuperação de um binário de uma aplicação.

Os protocolos acima descritos garantem a autenticidade e a integridade dos arqui-

vos executáveis submetidos à grade. Todas as mensagens trocadas entre o repositório de aplicações e seus clientes são devidamente assinadas e criptografadas. Pessoas mal intencionadas terão mais dificuldade ao utilizar técnicas de ataque conhecidas para modificar, interceptar ou fabricar dados, prejudicando assim o uso da grade. O uso de função de *hash* no sistema de arquivos é útil para tentar impedir a modificação das aplicações através de falhas de segurança no sistema de arquivos do gerenciador do aglomerado.

2.4.5. OurGrid

O OurGrid utiliza o MyGrid para implementar um sistema de Grade baseado numa rede *peer to peer*. O MyGrid é um sistema que teve como premissa de projeto contruir um sistema simplificado para executar aplicações sobre recursos computacionais distribuídos. No MyGrid o próprio usuário pode instalar um grade computacional com os recursos que dispõe. A instalação do MyGrid não requer nenhum privilégio especial de administrador.

O MyGrid define duas categorias de máquinas. A maquina base é ponto de acesso à Grade. Através da desta máquina, o usuário pode adicionar submeter e monitorar aplicações e ainda adicionar outras máquinas a Grade. As máquinas de *grid*, por sua vez, são as máquinas responsáveis pela execução de aplicação na Grade. As máquinas de *grid* e as máquinas base não necessitam compartilhar nenhum sistema de arquivo, bastando que sejam acessíveis pela máquina pelos usuários.

O MyGrid define a *Grid Machine Interface*, um conjunto mínimo de serviços que precisam estar disponíveis para que uma dada máquina possa ser adicionada ao Grid do usuário [Santos-Neto and Cirne, 2005]. Os serviços são transferências de arquivo, execução remota e interrupção de execução múltipla. O Mygrid provê as seguintes implementações para estes serviços:

1. *Grid Script* - usa aplicações básicas do sistema operacional;
2. *User Agent* - pequena aplicação desenvolvida em java;
3. Globus Proxy - direciona as operações necessárias para serviços implementados no Globus (GSI, GRAM e GridFTP).

O OurGrid é formado por três componentes básicos: MyGrid Broker, OurGrid *Peer* e um Sanboxing baseado no Xen denominada Swan. O MyGrid Broker provê um alto nível de abstração da grade. O Ourgrid é o componente responsável por gerenciar as máquinas que pertencem a um determinado domínio administrativo e obter acesso a máquinas em outros domínios [MyGrid/OurGrid, 2005]. E finalmente, o SWAN é uma solução de *sandboxing*[10] baseado na máquina virtual Xen*ho03:xen*.

2.4.5.1. Segurança

O OurGrid possui duas formas de autenticação: acesso direto e via OurGrid Peer. No acesso direto os usuários obtem os recursos pela máquina grid através de autenticação

[10]O *sandboxing* é uma tecnologia que cria ambientes de execução confinados para executar aplicações inseguras.

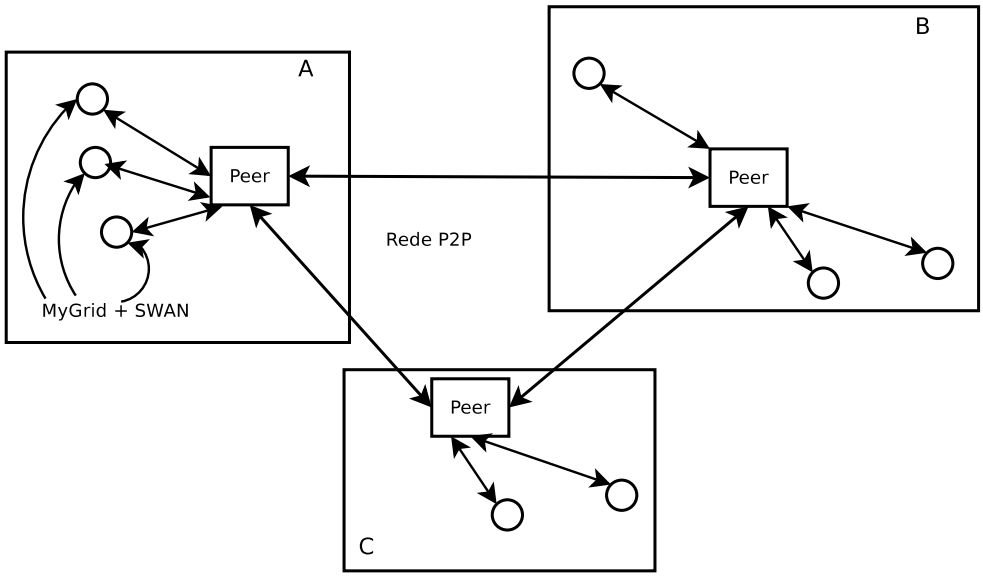

Figura 2.23. Arquitetura do OurGrid.

no sistema operacional via login e senha. A outra forma de autenticação é via certificados certificados digitais no formato X.509. Nesse modo de autenticação, somente módulos confiáveis podem comunicar entre si.

A autenticação do OurGrid é feita em duas fases [Santos-Neto and Cirne, 2005]. A primeira garante que o usuário tem permissão para solicitar serviços às maquinas grid. A segunda parte garante que o usuário não está solicitando serviços a uma falsa maquina grid. Dessa forma os usuários (através do broker), os OurGrid Peers gerenciam uma lista de certificados usadas para validar a tentativa de acesso.

O OurGrid utiliza os certificados para permitir comunicação segura entre o MyGrid Broker e o OurgridBroker. A segurança na comunicação é fornecida através do uso de RMI baseado em SSL (Secure Socket Layer), que garante comunicação criptografada.

Detsch [Detsch et al., 2004] propõe um arcabouço de segurança para redes P2P chamado P2PSLF (Peer-to-Peer Security Layer Framework) que foi implementado no OurGrid. Este arcabouço, ainda em desenvolvimento, define serviços de autenticação e confidencialidade para sistemas P2P. O P2PSFL foi implementado para ser completamente independente da aplicação do usuário. O arcabouço tem características bem interessantes como a modularidade, definição de requisitos de segurança por *peer* e redefinição dos requisitos dinamicamente.

A Figura 2.24 apresenta o resumo da arquitetura do P2PSLF. O arcabouço permite a definição de grupos de *peers* que compartilham requisitos de segurança. Cada *peer* possui perfis associados que representam os sentidos das mensagens entre os *peers* e os requisitos definidos. No exemplo da Figura 2.24, o *Peer* 1 exige autenticação quando houver troca de mensagens com os *Peers* 2 e 4. O *Peer* 1 também define que as mensagens com o *Peer* 3 serão autenticadas quando o sentido for do 1 para o 3 e autenticadas e confidenciais no sentido inverso. O *Peer* 3, por sua vez, envia mensagens autenticadas com confidencialidade para os *Peers* 1 e 4 e mensagens somente autenticadas no sentido

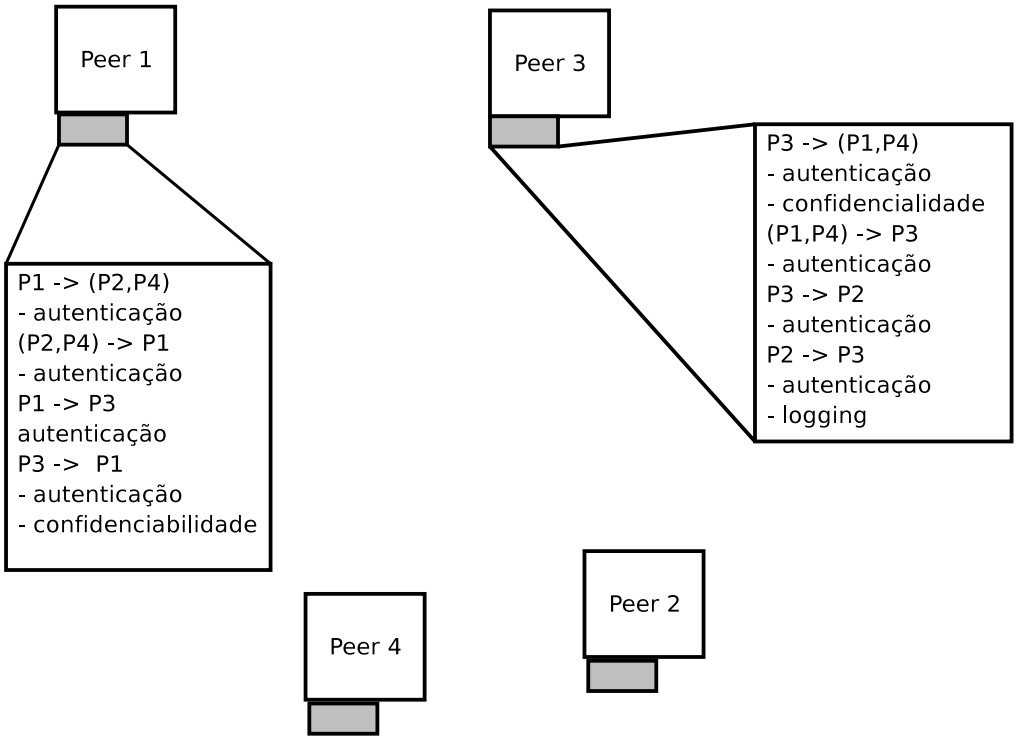

Figura 2.24. Arquitetura do P2PSFL.

contrário. E finalmente, o *Peer* 3 envia mensagens com autenticação para o *Peer* 2 e o mesmo *Peer* autentica e registra eventos ocorridos quando recebe mensagens do *Peer* 2.

Em cada um dos *peers* presentes na arquitetura há um módulo de configuração que é responsável pela descoberta de *peers* que formam o grupo. Assim que ingressa a rede, o módulo faz uma requisição especial de busca para descobrir as configurações de segurança dos *peers* remotos. Baseado nas informações devolvidas, o administrador pode reconfigurar os requisitos de segurança definidos para aquele nó.

2.5. Considerações finais

Neste capítulo fizemos uma breve introdução sobre segurança. Descrevemos os conceitos básicos de segurança e as suas ferramentas mais usadas. Apresentamos as Grades Computacionais, suas características, seus requisitos de segurança básicos e os mecanismos de segurança disponíveis para implementar seus requisitos. A seguir, levantamos o estado da arte em segurança de Grades Computacionais ao fazer um estudo de caso de implementações de segurança em grades computacionais.

As Grades Computacionais introduzem novos desafios na definição de soluções para seus requisitos de segurança. A característica distribuída e dispersa administrativamente requer a definição de mecanismos que sejam escaláveis, dinâmicos e confiáveis. A necessidade de que a autenticação e a autorização em uma grade seja feita de maneira separada direciona as soluções de segurança por um caminho diferente daquele necessário para tratar de sistemas convencionais.

As implementações de segurança nos sistemas de grades aqui apresentadas pos-

suem algumas caracteríticas em comum. Entre elas, a necessidade de criar implementações fléxiveis que permitam utilizar mecanismos diferentes de segurança. Sistemas como o Condor permitem que a autorização seja feita por mecanismos que vão desde uma infra-estrutura de chaves públicas até uma simples criação de um arquivo em um diretório temporário. O uso de *sandboxes*, protocolos padrão (como o SSL), utilização de delegação através de *proxies* são outros exemplos de caminhos compartilhados entre essas implementações de segurança.

2.5.1. Tendências futuras

Diversas tecnologias, tais como CORBA, JRMI e Serviços Web, surgiram ao longo dos últimos anos e poderiam ser usadas para padronizar uma infra-estrutura básica para grades computacionais e para segurança. Observamos, no entanto, que existe uma tendência (talvez pela própria caracteristica naturalmente heterogênia das grades computacionais) da convivência de padrões distintos. As grades terão que se adequar a essa realidade, sob a pena de ficarem isoladas, o que é um contra-senso pelos princípios definidos paras as grades computacionais.

O uso de *proxy* de certificados parece ser um bom caminho a ser seguido quando o problema for a delegação. O Globus parece ter norteado a questão com a definição da extensão dos certificados X.509. Eles apresentam uma boa solução para *login* único na rede, a delegação e a criação de identidades de uma maneira rápida e segura. Acreditamos no entanto, que a dependência ainda existente em um elemento central, a autoridade certificadora, ainda diminui a escalabilidade desta solução.

Alguns tipos de Grade podem requerer a autenticação de grupo. A autenticação de grupo permite que um conjunto de elementos compartilhem de forma segura chaves criptográficas. Dessa forma, é permitido que cada membro do Grupo possa obter privacidade em função do segredo que somente os membros do grupo conhecem. Essa tecnologia poderia ser usada par permitir, por exemplo, a transmissão de vídeos em difusão de forma segura.

Uma outra tendência a ser considerada são as federações. Uma federação é o agrupamento de elementos com afinidade. Um determinado congregado ao se filiar a uma federação, recebe uma identidade associada a esta federação. Com o processo de filiação, o congregado terá direitos a recursos disponíveis na federação a que pertence. Além de receber uma nova identidade, a federação facilita a administração da Grade.

O uso de *sandboxing* parece ser uma tendência como solução de isolamento de aplicações maliciosas em Grades Computacionais. A maioria dos sistemas de grade aqui apresentada utilizam a tecnologia de *sandboxing* para permitir que as aplicações sejam executads em um domínio de execução diferente. Esse isolamento, no entanto, é geralmente muito restritivo impedindo que as aplicações da grade tenham acessos a recursos. Além de tudo, por trás desta tecnololgia, geralmente há um ambiente que oferece uma sobrecarga considerável para os usuários (locais e da Grade).

Sabemos no entanto que não existe uma solução única e genérica para a segurança em Grades Computacionais. Na verdade até apostamos que sua característica dinâmica e flexível faça com que as Grades Computacionais sempre estejam em permanentemente

desenvolvimento. Sempre que houver uma nova tecnologia de segurança, as grades deverão se adequar a elas, sem ter que esquecer de implementar soluções para a vasta quantidade de mecanismos legados ainda em uso. Finalmente, o mundo acadêmico ainda tem uma trilha muito longa para percorrer na definição de sistemas que se adequem aos novos ambientes criados no contexto da Computação em Grade.

Referências

[Adams and Lloyd, 2002] Adams, C. and Lloyd, S. (2002). *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Andrade et al., 2003] Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. (2003). OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 61–86. Springer Verlag. Lect. Notes Comput. Sci. vol. 2862.

[Bellovin and Merritt, 1990] Bellovin, S. M. and Merritt, M. (1990). Limitations of the kerberos authentication system. *SIGCOMM Comput. Commun. Rev.*, 20(5):119–132.

[Berman et al., 2003] Berman, F., Fox, G., Hey, A. J. G., and Hey, T. (2003). *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, Inc.

[Booth et al., 2004] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). *Web Services Architecture*. World Wide Web Consortium.

[Clarke et al., 1999] Clarke, D., Elien, J.-E., Ellison, C., Fredette, M., Morcos, A., and Rivest, R. L. (1999). Certificate chain discovery in spki/sdsi. To be published, November 1999.

[Cohen, 1987] Cohen, F. (1987). Computer viruses: theory and experiments. *Comput. Secur.*, 6(1):22–35.

[de Camargo et al., 2004] de Camargo, R. Y., Goldchleger, A., Carneiro, M., and Kon, F. (2004). Grid: An Architectural Pattern. In *The 11th Conference on Pattern Languages of Programs (PLoP'2004)*, Monticello, Illinois, USA.

[Dierks and Allen, 1999] Dierks, T. and Allen, C. (1999). RFC 2246: The TLS protocol version 1. IETF RFC Publication. Status: PROPOSED STANDARD.

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

[Ellison et al., 1999] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Bell, S., and Ylonen, T. (1999). SPKI Certificate Theory. Internet RFC #2693.

[Epema et al., 1996] Epema, D., Livny, M., van Dantzig, R., Evers, X., and Pruyne, J. (1996). A worldwide flock of Condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12:53–65.

[Foster and Czajkowski, 2005] Foster, I. and Czajkowski, K. (2005). Modeling and managing state in distributed systems: the role of ogsi and wsrf. In *Proceedings of the IEEE*, volume 93, pages 604–612.

[Foster and Kesselman, 1997] Foster, I. and Kesselman, C. (1997). Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 2(11):115–128.

[Foster and Kesselman, 2003] Foster, I. and Kesselman, C. (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc.

[Foster et al., 2002] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Global Grid Forum, Open Grid Service Infrastructure Working Group.

[Foster et al., 1998] Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S. (1998). A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 83–92.

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of Supercomputer Applications*, 15(3):200–222.

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

[Garfinkel and Spafford, 1996] Garfinkel, S. and Spafford, G. (1996). *Practical UNIX & Internet Security*. O Reilly & Associates, Inc.

[Globus, 2004] Globus (2004). `http://www.globus.org`.

[Goldchleger, 2004] Goldchleger, A. (2004). Integrade: Um sistema de middleware para computação em grade oportunista. Tese de mestrado, IME/USP.

[Goldchleger et al., 2003] Goldchleger, A., Kon, F., vel Lejbman, A. G., and Finger, M. (2003). InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. In *Proceedings of the ACM/IFIP/USENIX Middleware'2003 1st International Workshop on Middleware for Grid Computing*, pages 232–234, Rio de Janeiro.

[Grafinkel and Spafford, 1996] Grafinkel, S. and Spafford, G. (1996). *Practical UNIX and Internet Security*. O'Reilly & Associates, Inc.

[Ian, 2005] Ian, V. W. (2005). Globus toolkit version 4 grid security infraestruture: A standards perspective. www-unix.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf.

[Kohl and Neuman, 1993] Kohl, J. and Neuman, C. (1993). The Kerberos network authentication service (v5). Internet RFC #1510.

[Krueger, 1988] Krueger, P. E. (1988). *Distributed scheduling for a changing environment*. PhD thesis, Madison, WI, USA.

[Lang and Schreiner, 2002] Lang, U. and Schreiner, R. (2002). *Developing Secure Distributed Systems with CORBA*. Artech House, Inc., Norwood, MA, USA.

[Lewis and Grimshaw, 1996] Lewis, M. J. and Grimshaw, A. (1996). The Core Legion Object Model. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing (HPDC '96)*, pages 551–561, Los Alamitos, California. IEEE Computer Society Press.

[Lhee and Chapin, 2003] Lhee, K.-S. and Chapin, S. J. (2003). Buffer overflow and format string overflow vulnerabilities. *Softw. Pract. Exper.*, 33(5):423–460.

[Menezes et al., 1996] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. (1996). *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA.

[MyGrid/OurGrid, 2005] MyGrid/OurGrid (2005). `http://www.ourgrid.org`.

[NIST SP-800-12, 1995] NIST SP-800-12 (1995). An introduction to computer security: The NIST handbook. Special Publication SP 800-12, National Institute of Standards and Technology (NIST).

[Pearlman et al., 2002] Pearlman, L., Welch, V., Foster, I., Kesselman, C., and Tuecke, S. (2002). A community authorization service for group collaboration. In *POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, pages 50–59, Washington, DC, USA. IEEE Computer Society.

[Ramachandran, 2002] Ramachandran, J. (2002). *Designing security architecture solutions*. John Wiley & Sons, Inc., New York, NY, USA.

[Rivest and Lampson, 1996] Rivest, R. L. and Lampson, B. (1996). SDSI – A simple distributed security infrastructure. Presented at CRYPTO'96 Rumpsession.

[Santos-Neto and Cirne, 2005] Santos-Neto, E. and Cirne, W. (2005). *Minicurso: Livro Texto*, chapter Grids Computacionais: Da Computação de Alto Desempenho a Serviços sob Demanda, pages 15–60. Sociedade Brasileira de Redes de Computadores.

[Sesame, 2003] Sesame (2003). Secure European System for Applications in a Multivendor Environment. https://www.cosic.esat.kuleuven.ac.be/sesame/.

[Sinha, 1996] Sinha, P. K. (1996). *Distributed Operating Systems: Concepts and Design*. Wiley-IEEE Press.

[Stallings, 2002] Stallings, W. (2002). *Network Security Essentials: Applications and Standards*. Prentice Hall Professional Technical Reference.

[Tanenbaum and Steen, 2002] Tanenbaum, A. S. and Steen, M. V. (2002). *Distributed Systems. Principles and Paradigms*. Prentince Hall.

[Terada, 2000] Terada, R. (2000). *Segurança de Dados. Criptografia em Redes de Computadores*. Editora Edgard Blücher Ltda.

[W3C, 2000] W3C (2000). *Simple Object Access Protocol (SOAP) 1.1*. URL: http://www.w3c.org/TR/SOAP.

[Welch et al., 2004] Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Tuecke, S., Gawor, J., Meder, S., and Siebenlist, F. (2004). X.509 proxy certificates for dynamic delegation. NIST Gaithersburg MD, USA.

[Xavier, 2004] Xavier, F. C. (2004). Um sistema de autorização baseado em uma infraestrutura de gerenciamento de privilégio. Tese de mestrado, IME/USP.

Capítulo 3

Serviços Distribuídos Tolerantes a Intrusões: resultados recentes e problemas abertos

Miguel Pupo Correia
LASIGE, Faculdade de Ciências da Universidade de Lisboa

Abstract

The idea of using dependability *concepts, mechanisms and architectures in the* security *domain is generating a lot of interest on both communities under the name of* intrusion tolerance. *Much of this attention has been created by the European MAFTIA project and the American OASIS program around 2000, although the notion comes from earlier. Albeit these projects have ended, much relevant work in the area has appeared in recent years, and now it is possible to know how to build* intrusion tolerant distributed services. *The objective is to have services with the properties of integrity, availability and confidentiality, even if some servers are attacked and controlled with success by hackers or malicious code. The chapter presents the state of the art in this area, clarifying the problems it solves and topics that remain open and have to be researched.*

Resumo

A idéia de aplicar conceitos, mecanismos e arquiteturas da área da confiança no funcionamento *no domínio da* segurança *tem gerado muito interesse em ambas as comunidades sob a designação de* tolerância a intrusões. *Muita da atenção foi criada pelo projeto europeu MAFTIA e pelo programa americano OASIS por volta do ano 2000, embora a noção venha de bem mais longe. Apesar desses projetos terem terminado, muito trabalho relevante tem surgido recentemente, sendo já possível ter idéias claras sobre como se podem concretizar* serviços distribuídos tolerantes a intrusões. *O objetivo consiste em garantir a integridade, disponibilidade e confidencialidade desses serviços mesmo que alguns servidores sejam atacados e controlados com sucesso por atacantes ou código nocivo. Este capítulo apresenta o estado da arte na área, clarificando os problemas que permite resolver e os tópicos que permanecem abertos e que precisam de ser pesquisados.*

3.1. Introdução

A *segurança* e a *confiança no funcionamento* (*security* e *dependability* em inglês) são duas áreas de pesquisa já com cerca de três décadas de existência. Quando se fala de *segurança*, está sempre presente a idéia de uma intenção maliciosa por parte de alguém, quer atuando diretamente, quer através da criação e distribuição de código nocivo (virus, vermes, etc.). O objetivo da *segurança* consiste em evitar que essa vontade de fazer mal prejudique dois tipos de bens: informação e serviços. Mais precisamente, as principais propriedades que a *segurança* pretende garantir são a confidencialidade, integridade e disponibilidade da informação e de serviços computacionais. Na *confiança no funcionamento*, que inclui a *tolerância a faltas*, a idéia subjacente é a de que o sistema se comporte de acordo com a sua especificação, perante os inúmeros problemas que podem ocorrer na prática: desde catástrofes naturais a *bugs* no software ...

Apesar de terem seguido caminhos diferentes ao longo destas três décadas, as duas áreas têm muito em comum. Na realidade, o objetivo das duas áreas é o mesmo: garantir que os sistemas computacionais funcionam corretamente. A ênfase da *segurança* tem sido em problemas de origem maliciosa (ataques, código nocivo), enquanto que o foco da *confiança no funcionamento* tem sido nos problemas de origem acidental. No entanto, as disciplinas não se excluem, pois a *segurança* pode tratar problemas de origem acidental e a *confiança no funcionamento* pode incluir problemas de origem maliciosa.

A *tolerância a intrusões (TI)* surge precisamente do encontro dessas duas áreas. Resumidamente, a idéia consiste em *aplicar o paradigma da tolerância a faltas no domínio de segurança*. Uma pequena estória pode ajudar a compreender em que consiste a TI:

> *O pirata pegou a luneta e observou o torreão da fortaleza. Era preciso conquistá-lo para chegar ao famoso tesouro do Rei daquele país, que tantos antes dele tinham cobiçado! O olho de vidro brilhou ... se não de felicidade pelo menos com o reflexo do sol.*
>
> *Os obstáculos para chegar ao torreão eram difíceis, mas não intransponíveis. Primeiro, teria de remar centenas de metros em plena noite tentando não ser vislumbrado. Depois iria acostar aos rochedos afiados contra os quais o mar se esmagava violentamente.*
>
> *Para descansar das manobras náuticas, um agradável exercício de escalada de 50 metros de escarpa rochosa, completado com a passagem da primeira muralha e dos seus guardas. Um fosso povoado por jacarés iria ser um bom momento para refrescar as idéias ... e a subida da parede da fortaleza iria dar-lhe tempo para secar a roupa.*
>
> *A tarefa não era simples, mas uma vez no torreão ...*

Se nesta pequena estória substituirmos "pirata" por "atacante", "fortaleza" por "sistema" e "tesouro" por informação ou serviço, percebemos que há um paralelo evidente com a *segurança* de sistemas computacionais. A abordagem clássica em *segurança* consiste em usar cuidadosamente todos os obstáculos da estória para dificultar a vida do pi-

rata: mar e rochedos, escarpa, muralha, guardas, jacarés ... que podem ser *firewalls*, controle de acesso, mecanismos biométricos, criptografia, redes privadas virtuais, etc. Passar por todos esses mecanismos é difícil, mas todos os dias muitos piratas em todo o mundo gritam de alegria quando chegam ao seu tesouro [CERT/CC, 2005, Turner et al., 2004].

A área da *confiança no funcionamento* tem uma abordagem algo diferente desta da *segurança*. Por exemplo, não basta usar as melhores técnicas de engenharia para que o computador do Airbus A380 não pare; é preciso ter vários computadores a bordo para *tolerar* esses eventos. A *tolerância a intrusões* pega nesta mesma idéia: não basta que o pirata tenha que ultrapassar obstáculos difíceis, embora essa dificuldade – todos os mecanismos clássicos da *segurança* – seja essencial. O que seria desejável é que o pirata tivesse que penetrar em vários torreões diferentes, em fortalezas diferentes, para conseguir pegar o tesouro!

O conceito de TI foi introduzido há já duas décadas por Fraga e Powell [Fraga and Powell, 1985][1]. No entanto, a TI começou a gerar maior interesse só mais recentemente, em parte devido ao projeto europeu MAFTIA[2] e ao programa americano OASIS [Lala, 2003][3], ambos iniciados por volta do ano 2000[4].

Apesar destes projetos terem terminado, muitos trabalhos relevantes têm surgido recentemente, sendo já possível ter idéias claras sobre como se podem concretizar *serviços distribuídos tolerantes a intrusões*. O objetivo consiste em garantir a integridade, disponibilidade e confidencialidade de serviços constituídos por diversos servidores ligados através de uma rede, mesmo que alguns desses servidores sejam atacados e controlados com sucesso por atacantes (*hackers*, *crackers*) ou por código nocivo (virus, vermes, etc.). Alguns exemplos de serviços que podem assim se tornar seguros são PKIs, sistemas de arquivos distribuídos, comércio eletrônico ou serviços de autorização. A TI não se reduz aos serviços distribuídos, mas essa é provavelmente a área onde os trabalhos mais relevantes têm sido realizados.

O objetivo deste capítulo consiste em apresentar o estado da arte na área, clarificando os problemas que esta permite resolver e os tópicos que permanecem abertos e que precisam de ser pesquisados.

O capítulo está organizado da seguinte forma. A seção 3.2 apresenta os principais conceitos de TI e a sua relação com a *segurança*. A seção 3.3 apresenta as soluções e algoritmos para concretizar serviços distribuídos TI com replicação. Usando estas técnicas – que incluem a replicação de máquinas de estados e os quoruns – consegue-se aumentar a disponibilidade e a integridade desses serviços. A seção 3.4 introduz as abordagens que permitem garantir também a confidencialidade dos dados usando fragmentação. A seção 3.5 explica algumas técnicas que permitem processar as intrusões dos servidores, aumentando assim o número de intrusões que é possível tolerar. A seção 3.6 passa dos algoritmos para a arquitetura de serviços distribuídos TI, introduzindo diversas arquiteturas e sistemas propostos na bibliografia. Finalmente, a seção 3.7 apresenta os principais

[1] Logo podemos dizer que tem sangue brasileiro ...

[2] http://www.maftia.org

[3] http://www.tolerantsystems.org

[4] O termo *survivability* tem também sido usado para apelidar alguns trabalhos na área, sobretudo quando a origem é americana.

problemas abertos da área, procurando dar pistas sobre a pesquisa que é preciso fazer. A seção 3.8 apresenta algumas conclusões.

3.2. Conceitos básicos de Tolerância a Intrusões

Esta seção explica as bases da TI, começando por uma introdução à *confiança no funcionamento* [Avizienis et al., 2004, Veríssimo and Rodrigues, 2001]. Em relação a esta área seguiremos a terminologia para português de Veríssimo e de Lemos, na variação brasileira quando existirem duas versões de um termo [Veríssimo and de Lemos, 1989].

3.2.1. Confiança no funcionamento

Um *sistema* é uma entidade que interage com outros sistemas – computacionais, mecânicos, físicos, seres humanos – através da sua *fronteira*. Tudo o que está fora do sistema constitui o seu *ambiente*. Um sistema computacional contém diversos *componentes* e é caracterizado pelo que faz – a sua *funcionalidade* – e por um conjunto de propriedades *não funcionais*, como o seu desempenho, a sua *segurança*, a sua confiabilidade, etc. Um sistema fornece um determinado *serviço*, através de uma *interface*, a um *utilizador*, e tem um *estado* que muda com o passar do tempo.

Um sistema fornece um serviço *correto* se este obedece à especificação do sistema. Caso contrário existe uma *falha* do serviço. O objetivo da *confiança no funcionamento* consiste em fazer com que o serviço permaneça correto, ou seja, que não falhe. Para que isso seja possível é preciso entender o processo que leva à falha.

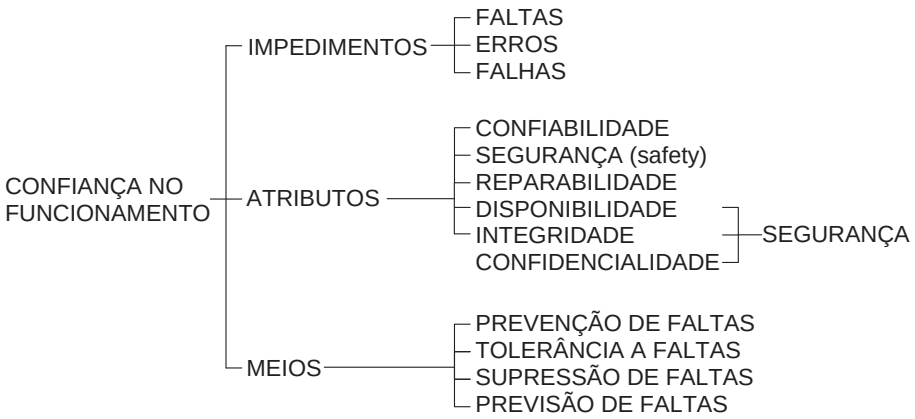

Figura 3.1. Conceitos de *confiança no funcionamento* [Avizienis et al., 2004].

Os impedimentos à *confiança no funcionamento* assumem três facetas (v. fig. 3.1): falta, erro e falha (já referida). Uma *falta* é a causa remota de uma falha. Uma falta pode ser interna (p. ex. um defeito na memória RAM) ou externa (p. ex. um operador que tropeça e desliga um cabo). Um *erro* é a consequência de uma falta no estado do sistema (p. ex. um registo corrompido por ter sido lido da memória RAM defeituosa). Uma falta pode ficar *dormente*, pode não gerar imediatamente um erro (quando gera diz-se *ativa*); um erro pode ou não gerar a falha do sistema (p. ex. se o registo não for lido o sistema não falha devido a esse erro). Uma falta interna corresponde à falha de um componente do sistema. Se olharmos para esse componente como um sistema, também a sua falha pode

ser causado pela falha de um dos seus componentes, podendo existir uma sequência que conduza à falha do sistema:

falta → erro → falha → falta → erro → falha ...

A *confiança no funcionamento* pretende garantir um conjunto de *atributos*: *confiabilidade* (continuidade do serviço correto); *segurança (safety)* (ausência de consequências catastróficas sobre os utilizadores); *reparabilidade* (capacidade de receber modificações e reparações); *disponibilidade* (prontidão do serviço correto); *integridade* (ausência de alterações inadequadas ao sistema). É evidente que os dois últimos atributos coincidem com propriedades básicas da *segurança*. Uma propriedade de *segurança* geralmente não considerada no âmbito da *confiança no funcionamento* é a *confidencialidade*, a ausência de revelação inadequada de informação.

Uma questão especialmente importante é a dos *meios* para procurar garantir a *confiança no funcionamento*. Os meios são muitos, fruto de muitos anos de pesquisa, mas podem ser agrupados em quatro categorias:

- *Prevenção de faltas* - meios para prevenir a introdução de faltas. Faz parte do processo normal da engenharia de sistemas, tanto da engenharia de software como da de hardware.
- *Tolerância a faltas* - meios para evitar a falha do serviço quando ocorrem faltas. Os meios desta categoria podem ser divididos em duas sub-categorias:
  - *mascaramento de faltas* - usar redundância para garantir que as faltas não causem a falha do sistema;
  - *detecção e processamento* - detectar a ocorrência de erros e processá-los de forma a os neutralizar.
- *Supressão de faltas* - meios usados durante o projeto do sistema para reduzir o número e/ou a severidade das faltas. Estes meios incluem diversas técnicas de verificação e validação de sistemas, tanto de hardware como de software.
- *Previsão de faltas* - meios para estimar o número de faltas no sistema, e prever o número e consequências de faltas futuras. As principais técnicas podem ser divididas em modelagem e teste.

A *confiança no funcionamento* é consequência da combinação eficaz destes meios. Os mecanismos clássicos de *segurança*, como o controle de acesso e a autenticação, poderiam ser englobados nos meios para *prevenção de faltas*. Já a TI, tenta usar os mecanismos e conceitos da *tolerância a faltas* no domínio da *segurança*. A *supressão de faltas* e a *previsão de faltas* são ortogonais aos outros meios e não vão ser mais considerados.

3.2.2. Tolerância a intrusões

Depois do que foi dito sobre *confiança no funcionamento* conclui-se facilmente em que consiste aplicar o paradigma da *tolerância a faltas* no domínio da *segurança* – a *tolerância a intrusões* [Veríssimo et al., 2003, Adelsbach et al., 2002]:

- assumir e aceitar que o sistema permanece sempre mais ou menos vulnerável;
- assumir e aceitar que os componentes do sistema podem ser atacados e que alguns desses ataques terão sucesso;
- garantir que o sistema como um todo permanece seguro e operacional, ou seja, que não falha.

O nome *tolerância a intrusões* dá a entender que *intrusões* são faltas. Na realidade, não só as intrusões mas também as vulnerabilidades e os ataques são faltas. Uma *vulnerabilidade* é uma falta de projeto ou de configuração, geralmente acidental (i.e., não intencional), que pode ser explorada com fins maliciosos. Um *ataque* é uma falta intencional, maliciosa, que visa explorar uma ou mais vulnerabilidades. Uma *intrusão* é o resultado de um ataque que tem sucesso em explorar uma ou mais vulnerabilidades.

Como já foi dito, no domínio da *confiança no funcionamento* geralmente são consideradas apenas faltas acidentais. Muitas vezes são tratadas apenas as mais simples entre essas faltas, as faltas de parada (p. ex. o *crash* de processos ou máquinas). No domínio da *segurança* geralmente é irrealista, logo perigoso, levantar hipóteses sobre o modo como um componente falha. Por isso, as faltas maliciosas são geralmente consideradas como podendo ser de qualquer tipo, logo sendo englobadas na categoria de faltas mais geral: as *faltas arbitrárias*, também denominadas de *faltas bizantinas*[5]. Em *tolerância a intrusões* os termos *intrusão* e *falta bizantina* são usados geralmente como sinônimos.

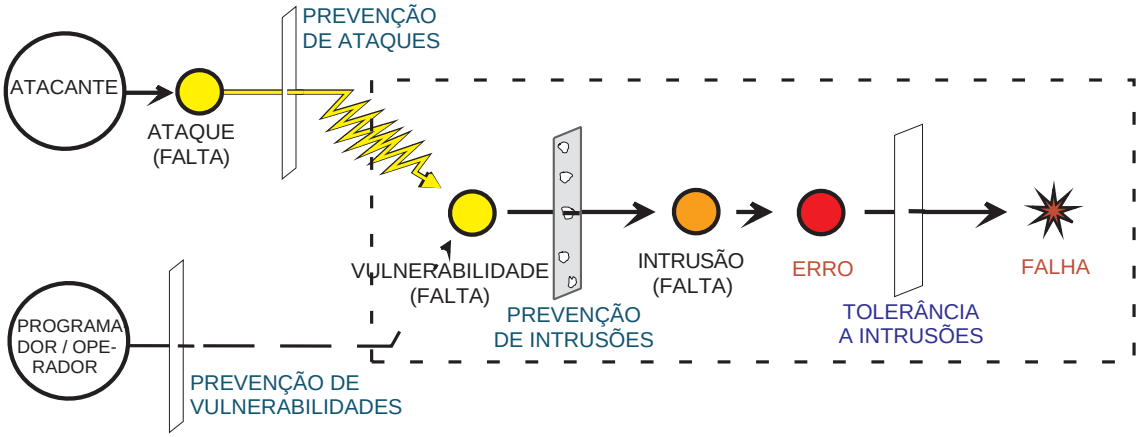

Figura 3.2. O modelo AVI e os mecanismos para evitar a falha [Veríssimo et al., 2003].

Esta relação entre as noções de ataque-vulnerabilidade-intrusão e falta não deve ser descartada como uma questão de nomenclatura. Pelo contrário, serve para entender o processo de falha de um sistema e, consequentemente, os mecanismos que se podem usar para evitar que isso aconteça. Essa relação é ilustrada pela figura 3.2, que é auto-explicativa.

Em *segurança*, a noção de *trustworthiness* diz a que ponto um componente ou sistema satisfaz um conjunto de propriedades (de segurança). Se generalizarmos esta

[5]A denominação *faltas bizantinas* vem de um artigo clássico que apresenta um protocolo tolerante a faltas maliciosas através de um problema envolvendo generais bizantinos [Lamport et al., 1982].

noção para incluir os outros *atributos* da figura 3.1, então *trustworthiness* torna-se um sinônimo de *confiança no funcionamento* (*dependability*). Outra noção relacionada é a de confiança (*trust*): a dependência de um componente em relação às propriedades de segurança de outro componente. Também esta noção pode ser estendida para incluir as outras propriedades de *confiança no funcionamento*.

Na seção anterior falámos dos *meios* para obter *confiança no funcionamento*. O projeto de *serviços distribuídos tolerantes a intrusões* baseia-se numa conjugação dos quatro meios. Em relação à *tolerância a faltas*, a maior parte das soluções que veremos no capítulo são baseadas em *mascaramento de faltas*. Os serviços baseados nesses tipo de mecanismos *mascaram*, ou escondem, a existência de faltas. Para isso, usam-se não um mas vários servidores – redundância – em conjunto com protocolos de comunicação que permitam fazer esse mascaramento (v. fig. 3.3). Esses *protocolos* (ou *algoritmos distribuídos*) têm também de ser tolerantes a intrusões. Os serviços TI podem também usar o outro tipo de mecanismos de *tolerância a faltas*, o *processamento de erros*, para remover as intrusões que ocorram e assim evitar que o sistema falhe.

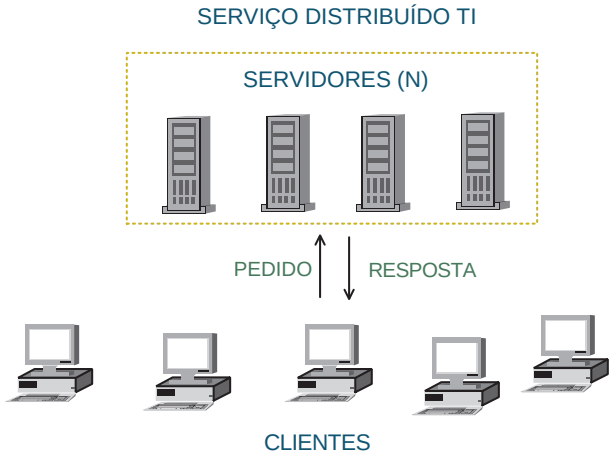

Figura 3.3. Arquitetura genérica de um sistema com um *serviço distribuído tolerante a intrusões*.

3.2.3. Sistemas distribuídos

Como o tema do capítulo é a TI em *sistemas distribuídos*, é importante fazer uma breve introdução a este tema. A definição clássica de Lamport diz que *um sistema distribuído é aquele que não o deixa trabalhar por causa da falha de um computador do qual nunca ouviu falar*. Considerando essa definição, própria de um investigador em *tolerância a faltas*, é importante apontar que o comportamento dos sistemas distribuídos é complexo, logo para raciocinar sobre esses sistemas são usados *modelos*, como em qualquer outro ramo da ciência.

O *modelo topológico* diz como as máquinas são interligadas por uma rede. Todos os trabalhos que vamos abordar usam um modelo topológico simples com conectividade total: todas as máquinas têm uma canal de comunicação com todas as outras[6].

[6]Na realidade algumas arquiteturas mais complexas que veremos na seção 3.6 dividem a rede em segmentos independentes e filtram o tráfego que passa entre eles. No entanto os algoritmos distribuídos são

O *modelo de falhas* define hipóteses sobre como podem falhar os componentes do sistema. Nos trabalhos que vamos analisar geralmente consideram-se faltas bizantinas, como já apontado, embora por vezes sejam usados determinados mecanismos para excluir *a priori* certos tipos de faltas (p. ex. usando canais SSL/TLS podem-se excluir as faltas na rede, exceto a quebra total de comunicação). Os modelos híbridos consideram diferentes hipóteses de faltas sobre diferentes partes do sistema, assumindo por exemplo a existência de certos componentes simples seguros [Veríssimo et al., 2000, Correia et al., 2002a].

O *modelo temporal* consiste num conjunto de hipóteses sobre o comportamento do sistema em termos de tempo. Os trabalhos que vamos estudar geralmente consideram o *modelo assíncrono*, que não assume qualquer hipótese sobre os tempos de processamento e de comunicação no sistema[7]. Este modelo é escolhido por prudência, digamos assim, pois muitas hipóteses temporais podem ser quebradas através de certos ataques (p. ex. uma hipótese sobre o atraso de comunicação pode ser quebrada através de um ataque de negação de serviço). No entanto, esta questão é algo mais complicada. Um problema importante em sistemas distribuídos é chamado de *consenso*. O problema pode ser formulado informalmente da seguinte forma: dado um conjunto de *processos*[8] cada um com um valor inicial; como fazer com que todos os processos corretos (ou seja, que não falhem) cheguem a acordo sobre um único valor? O problema parece simples, mas na realidade foi provado que não tem solução determinista num sistema assíncrono nem que apenas um processo possa falhar por parada [Fischer et al., 1985]. Esse resultado (FLP) tem consequências em inúmeros problemas de sistemas distribuídos que são equivalentes ao consenso, por exemplo, a entrega de mensagens com ordem total ou a comunicação em grupo com sincronia de vistas. Para contornar este resultado é preciso usar protocolos aleatórios [Ben-Or, 1983, Rabin, 1983], detectores de falhas [Chandra and Toueg, 1996] ou outras técnicas. Algumas destas técnicas escondem hipóteses temporais que podem introduzir vulnerabilidades.

3.2.4. Criptografia de limiar

O termo *criptografia de limiar* denomina um conjunto de algoritmos tipicamente de tolerância a faltas/intrusões mas que surgiram no âmbito da *segurança*[9]. Esses algoritmos por vezes constituem componentes importantes dos trabalhos que vamos apresentar mais à frente, logo é importante introduzi-los desde já. A bibliografia sobre o tema é extensa, mas um bom resumo encontra-se em [Gemmell, 1997].

A criptografia de limiar assume duas formas básicas. Sejam dados N processos, cada um detendo uma determinada *parte* secreta. O objetivo de um algoritmo de *partilha*

executados separadamente em cada segmento, logo o modelo de conectividade total continua a aplicar-se.

[7]No outro extremo do espetro dos modelos temporais está o *modelo síncrono*, que assume limites de tempo de processamento e comunicação. Entre os dois extremos há diversos modelos intermédios, geralmente chamados de modelos de sincronia parcial [Dwork et al., 1988].

[8]Ao longo de todo o capítulo vamos usar o termo *processo* para significar uma entidade que participa num algoritmo ou protocolo. Alguns termos usados com o mesmo significado são: processador, participante, parte, jogador. Um processo no qual tenha ocorrido uma intrusão e se desvie do comportamento especificado diz-se *malicioso*. Caso contrário diz-se *correto*.

[9]Nesta seção, como em todo o capítulo, vamos assumir que o adversário ou atacante – a entidade que procura quebrar o funcionamento de um serviço ou protocolo – é limitado computacionalmente, ou seja, que não consegue quebrar as primitivas criptográficas usadas.

de segredos é permitir que k processos combinem as suas partes e revelem determinado segredo s, garantindo simultaneamente que um conluio de até $k-1$ processos maliciosos não consegue fazer outro tanto, nem sequer obter qualquer informação relevante sobre s. Um algoritmo de *partilha de funções* permite que k processos apliquem determinada função F, não sendo possível até $k-1$ processos fazerem o mesmo. Um tipo especialmente importante de algoritmos deste último tipo são os algoritmos de *assinatura de limiar*, que permitem a um conjunto de processos criar uma assinatura criptográfica sem revelar a chave privada. No entanto, convém notar que um algoritmo de criptografia de limiar pode ser substituído por vetores com assinaturas, p. ex. RSA, uma por cada processo, inclusive com melhor desempenho [Cachin, 2002].

O algoritmo original de partilha de segredos de Shamir pode ajudar a entender como funciona a criptografia de limiar [Shamir, 1979] (este resultado foi desenvolvido ao mesmo tempo que outro semelhante de Blakley [Blakley, 1979]). O esquema é baseado em duas propriedades dos polinômios:

- dados quaisquer $d+1$ pontos distintos da curva definida pelo polinômio, é possível determinar qualquer outro ponto do polinômio;
- se os índices a_i do polinômio forem todos desconhecidos, o conhecimento de até d pontos da curva não revela nenhuma informação sobre outros pontos.

Seja dado um *polinômio* de grau d:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d \tag{1}$$

O algoritmo de Shamir considera a existência de um processo que pretende guardar o segredo s. Esse processo define um polinômio de grau $d = k-1$ de índices a_i aleatórios, exceto $a_0 = p(0) = s$. Depois, calcula $p(x)$ para N valores diferentes e aleatórios de x e distribui cada uma dessas *partes* por um dos N processos. Atendendo às duas propriedades dos polinômios acima, o segredo pode ser reconstituído por k processos mas não por $k-1$. A reconstituição é feita usando a interpolação de Lagrange:

$$p(0) = \sum_{i=1}^{k} \left(p(x_i) \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \right) \tag{2}$$

Esta é a idéia básica do funcionamento do algoritmo de Shamir. A única simplificação feita é a de que na prática não se podem usar valores arbitrariamente grandes, logo o algoritmo considera um número primo grande e faz todos os cálculos *módulo* esse número (ou seja, usa sempre o resto da divisão do resultado por esse número).

O algoritmo de Shamir tem duas limitações. A primeira é a de que um processo não tem como saber se a sua parte é "boa", ou seja, se combinada com outras $k-1$ partes boas reconstrói o segredo. Essa lacuna foi preenchida mais tarde por algoritmos de *partilha de segredos verificável*. A segunda limitação é que se uma das partes usadas para reconstituir o segredo estiver corrompida, p. ex. por ser fornecida por um processo

malicioso, não é reconstituído o verdadeiro segredo e pode nem sequer ser possível compreender que este está errado. Para resolver este problema foram desenvolvidos algoritmos *robustos*, baseados em *provas de conhecimento zero*. Existem ainda algoritmos de criptografia de limiar *proativos* mas isso é um tema para a seção 3.5.

3.3. Replicação: garantindo disponibilidade e integridade

A idéia básica da replicação consiste em distribuir cópias do código e dos dados de determinado serviço por um conjunto de servidores. A replicação tem sido amplamente usada em *tolerância a faltas* para garantir a disponibilidade e a confiabilidade de serviços distribuídos. Muitos dos trabalhos em *serviços distribuídos TI* são também baseado em replicação. Este tipo de soluções permite garantir a *disponibilidade* e a *integridade* do serviço se houver intrusões num número limitado de réplicas, geralmente menos de um terço.

Voltando à nossa estória do pirata, a idéia consiste em ter diversos torreões. O tesouro não é propriamente monetário (ouro, jóias ...) mas um serviço fornecido pelo sistema de fortalezas, digamos um serviço de vigia da entrada no único porto daquele país. O objetivo do pirata é interromper o serviço de informações, ou fazê-lo dar informação errada de modo a que os navios embandeirados pelo crânio e as tíbias possam entrar no porto. Para isso terá de invadir *vários torreões*, caso contrário o serviço fornecido pelo sistema de fortalezas permanecerá íntegro e disponível.

Os principais trabalhos nesta área podem ser classificados como os que fazem replicação de máquinas de estados [Lamport, 1978, Schneider, 1990] e os que usam quoruns bizantinos [Malkhi and Reiter, 1998a].

3.3.1. Replicação de máquinas de estados

A *replicação de máquinas de estados (RME)* é uma solução genérica para a concretização de *serviços* tolerantes a faltas [Schneider, 1990]. Um serviço oferece um conjunto de *operações* aos seus *clientes*, que os invocam através de *pedidos*. Um serviço é concretizado através de um conjunto de *N servidores* $s_i \in U$ (também chamados de *réplicas* neste contexto)[10]. A figura 3.3 ilustra estes conceitos.

Cada servidor é uma *máquina de estados*, definida por *variáveis de estado* que definem o seu estado, e por *comandos* que modificam esse estado. Os comandos têm de ser atômicos, ou seja, não podem interferir uns com os outros. Todos os servidores seguem a mesma sequência de estados, para o que é suficiente satisfazer quatro propriedades:

- *Estado inicial.* Todos os servidores começam no mesmo estado.
- *Acordo.* Todos os servidores executam os mesmos comandos.
- *Ordem total.* Todos os servidores executam os comandos pela mesma ordem.
- *Determinismo.* O mesmo comando executado no mesmo estado inicial gera o mesmo estado final.

[10] A nomenclatura usada nos diversos trabalhos varia bastante. Neste capítulo usaremos uma nomenclatura coerente.

A primeira propriedade é geralmente simples de garantir. A segunda e a terceira podem ser forçadas usando um *protocolo de difusão atômica* (ou de difusão com ordem total). Quanto à quarta propriedade, vamos considerá-a uma premissa por ora mas voltaremos a ela na seção 3.7.

Um parâmetro importante quando de fala de um serviço tolerante a faltas/intrusões é a *resistência* (*resilience*), o número máximo de servidores que podem falhar para o serviço se manter correto. Em sistemas assíncronos TI baseados em replicação de máquinas de estado este limite é imposto pelo protocolo de difusão atômica, cuja resistência máxima é de $f = \lfloor \frac{N-1}{3} \rfloor$ em N servidores [Bracha and Toueg, 1985, Hadzilacos and Toueg, 1994, Cachin et al., 2001]. Uma forma mais clara de dizer o mesmo, que por isso é a que vamos usar ao longo do texto, é a de que são necessários (pelo menos) $3f + 1$ servidores para tolerar f servidores que falham. O problema da difusão atômica é, como já foi referido, equivalente ao do consenso [Cachin et al., 2001]. Diversas soluções têm sido apresentadas na bibliografia para resolver consenso tolerante a faltas bizantinas [Ben-Or, 1983, Rabin, 1983, Bracha and Toueg, 1985, Malkhi and Reiter, 1997b, Doudou and Schiper, 1997, Cachin et al., 2000, Baldoni et al., 2000, Doudou et al., 2002, Kihlstrom et al., 2003, Correia et al., 2005a, Neves et al., 2005], como aliás também para difusão atômica [Reiter, 1994, Moser and Melliar-Smith, 1999, Kihlstrom et al., 2001, Castro and Liskov, 2002, Cachin and Poritz, 2002].

3.3.1.1. BFT

Um algoritmo ideal para aprofundar o tema da RME tolerante a intrusões é o BFT (*Byzantine Fault Tolerance*), um dos mais citados trabalhos de TI [Castro and Liskov, 2002]. Este algoritmo tem a resistência máxima possível em sistemas assíncronos: $N \geq 3f + 1$. Na discussão que se segue vamos considerar o caso mais restrito: $N = 3f + 1$[11].

O grande objetivo visado no BFT foi ter um protocolo correto e com bom desempenho. Em relação à correção, o BFT satisfaz sempre as suas propriedades de segurança mas o progresso do algoritmo, nomeadamente quando há mudança de vistas, depende de uma hipótese temporal fraca: o atraso na rede não cresce exponencialmente. Quanto ao desempenho, um sistema de arquivos distribuído NFS TI baseado na biblioteca BFT teve um desempenho entre 2% melhor e 24% pior do que soluções não seguras nem replicadas.

A principal opção que tornou o bom desempenho possível foi evitar o uso de criptografia de chave pública [Diffie and Hellman, 1976, Rivest et al., 1978] durante o funcionamento normal do sistema, ou seja, quando não há intrusões. As mensagens são assinadas usando *message authentication codes*, MACs [Menezes et al., 1997]. Cada par cliente-servidor partilha uma chave secreta. Cada mensagem ponto-a-ponto leva um MAC calculado com a chave partilhada pelo emissor/destinatário e cada mensagem por difusão leva um vetor com um MAC calculado com a chave compartilhada pelo emissor e cada destinatário. Com os servidores passa-se o mesmo, mas cada par de servidores partilha duas chaves secretas, uma para a comunicação em cada direção.

[11] Este é o caso mais restrito pois podia-se também ter $N = 3f + 2$, $N = 3f + 3$, etc.

Quando um cliente pretende enviar um pedido ao serviço, difunde uma mensagem com o comando, uma estampilha temporal (*timestamp*) e um vetor de MACs para todos os servidores. A estampilha serve para garantir que cada pedido de um cliente é executado precisamente uma vez. Cada servidor processa a mensagem (aliás, qualquer mensagem) apenas se o MAC que lhe corresponde estiver correto. Também um cliente só processa mensagens com MAC correto. O cliente aceita a resposta ao seu pedido quando recebe $f+1$ cópias vindas de diferentes servidores, o que garante que pelo menos uma das cópias vem de um servidor correto (parte-se da hipótese de que no máximo f sofrem intrusões). Se a resposta não chega, o cliente retransmite o pedido.

O algoritmo é baseado numa mistura de *replicação passiva* (ou *primário-secundário*) e *replicação ativa*. As réplicas vão mudando de configuração, sendo cada uma das configurações denominada uma *vista*. Em cada vista um servidor é o *primário* e os restantes são os *secundários*. A ordem de execução dos pedidos é definida pelo primário, atribuindo-lhe o próximo número de sequência a cada pedido recebido e reenviando-o para os secundários. Se o primário for malicioso, pode dar o mesmo número a dois pedidos, parar de atribuir números, ou deixar intervalos entre os números. Por isso, os secundários verificam os números de sequência atribuídos pelo primário e marcam o tempo para ver se ele pára. Quando os secundários suspeitam que o primário falhou, mudam de vista, logo de primário.

O protocolo de difusão atômica em operação normal tem três fases: *pre-prepare*, *prepare* e *commit* (v. fig. 3.4). As duas primeiras servem para ordenar pedidos enviados numa mesma vista, mesmo que o primário seja malicioso. A terceira fase garante a ordenação dos pedidos entre vistas diferentes.

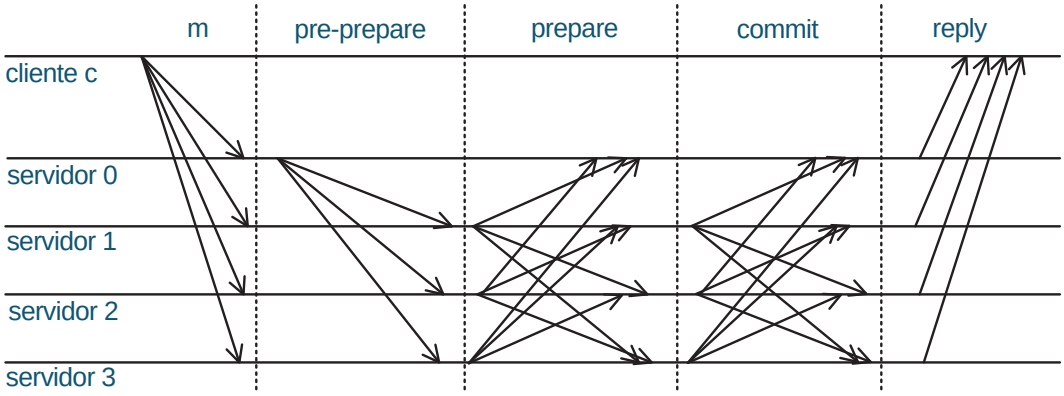

Figura 3.4. Algoritmo BFT em funcionamento normal [Castro and Liskov, 2002].

Na fase *pre-prepare* o primário (*servidor* 0 na figura) difunde o pedido recebido, com um número de sequência n e o número da vista v, para todos os secundários. Um secundário aceita esta mensagem se estiver na vista v e não tiver aceitado outra mensagem *pre-prepare* com os mesmos v e n. Caso aceite a mensagem e tenha recebido esse mesmo pedido do cliente, difunde uma mensagem *prepare* com um resumo criptográfico (*hash* [Menezes et al., 1997]) da mensagem para todas as réplicas (caso contrário não faz nada). Desta forma a réplica aceita atribuir o número de sequência n a esse pedido. Quando um servidor receber $2f$ mensagens *prepare* de outras réplicas, o pedido diz-se

preparado.

Este esquema força diversas propriedades importantes *numa mesma vista*. Primeiro, um primário malicioso não pode "criar" um pedido do nada pois as réplicas corretas só processam pedidos que tenham recebido de um cliente. Segundo, um cliente ou um primário maliciosos não podem provocar a execução de dois comandos diferentes pela mesma ordem em duas réplicas corretas diferentes. A justificativa é a seguinte. Uma réplica correta só executa o comando se receber $2f+1$ mensagens *prepare* (contando com a sua), logo pelo menos $f+1$ dessas mensagens vêm de réplicas corretas. Mesmo que todas as réplicas maliciosas – no máximo f – enviem mensagens diferentes com o mesmo número a réplicas diferentes, isso não é suficiente para fazer dois quoruns de $2f+1$ mensagens: $2(2f+1) > 3f+1$.

Este esquema garante a ordem dentro de uma vista mas não quando há mudança de vista. A fase de *commit* resolve esse problema. Quando uma réplica tem um pedido preparado, difunde uma mensagem *commit* para as outras réplicas. Quando uma réplica tem $2f+1$ mensagens *commit* então o pedido diz-se confirmado e pode ser executado. Se a vista mudar o novo primário têm de propagar os pedidos confirmados para a vista seguinte. Uma réplica executa um comando num pedido quando este tiver sido confirmado e todos os comandos em pedidos com número de ordem inferior tiverem sido executados. Quando o comando termina é enviada uma mensagem *reply* ao cliente que o solicitou.

Como já vimos, para garantir que o serviço continua a funcionar se o primário falhar é preciso mudar de vista. Vimos acima que os secundários podem suspeitar que o primário falhou. Fala-se em *suspeitar* em lugar de *detectar* pois em sistemas assíncronos pode existir incerteza em relação a essa falha. O modelo assíncrono não impõe limites temporais para a comunicação e processamento, logo o fato de o primário não responder durante algum tempo, não significa que falhou: ele ou a comunicação podem estar simplesmente lentos. Esse, aliás, é o ponto fraco do BFT: um atacante pode atrasar o sistema atrasando a comunicação e forçando a mudança constante de primário. O protocolo de mudança de vista é apresentado esquematicamente na figura 3.5. A questão crucial é garantir que os pedidos já confirmados são processados na nova vista.

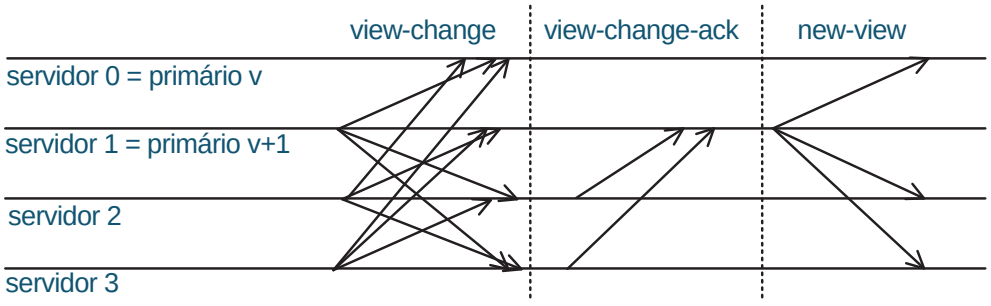

Figura 3.5. Mudança de vista no BFT [Castro and Liskov, 2002].

O bom desempenho do BFT deve-se não apenas ao uso de criptografia simétrica mas também a diversas otimizações. Duas delas merecem ser mencionadas. Para garantir a propriedade de ordem total da RME é essencial ordenar as escritas mas não as leituras.

Assim, o protocolo de leitura pode ser simplificado: o cliente envia um pedido de leitura para todas as réplicas e estas devolvem-lhe o valor pedido. Depois o cliente tenta recolher $f+1$ respostas idênticas. Se isso acontecer, a operação terminou. Caso haja escritas concorrentes e não seja possível recolher respostas idênticas, o cliente volta a fazer o pedido usando o protocolo que vimos atrás. No entanto, é fácil compreender que a probabilidade deste segundo caso acontecer é muito baixa.

A segunda otimização consiste em fazer processamento em pacotes de pedidos (*batching*). Em lugar de o primário enviar cada pedido recebido usando o protocolo explicado acima, tenta juntar diversos pedidos num só e enviá-lo.

Muito mais poderia ser dito sobre o BFT. As réplicas têm de guardar diversos registos (*logs*) que têm de ser limpos para evitar que cresçam indefinidamente. O BFT resolve a questão com uma espécie de *garbage collection* baseada em provas de que a informação sobre uma mensagem pode ser descartada. Uma versão do BFT, BFT-PR, utiliza recuperação proativa para processar as intrusões em servidores, uma questão para a seção 3.5.

Um artigo recente apresenta uma solução para RME TI com menor número de passos do que o BFT [Martin and Alvisi, 2005]. No entanto, isso é feito à custa de piorar a resistência para $N \geq 5f+1$.

3.3.1.2. Rampart

O sistema Rampart é um sistema que suporta RME TI que surgiu antes do BFT [Reiter, 1995, Reiter, 1994]. Na realidade o Rampart é mais genérico do que o BFT, pois é um sistema de comunicação em grupo que oferece primitivas de comunicação como difusão fiável e difusão atômica com sincronia de vistas. Tem um protocolo de filiação (*membership*) que permite a entrada e a saída de membros num grupo e a remoção de membros maliciosos, logo o conjunto de servidores que concretizam um serviço não é fixo. Neste contexto a *vista* é o conjunto de membros do grupo num determinado momento. No âmbito do projeto ITUA do programa OASIS foi feita uma concretização completa do sistema [Ramasamy et al., 2002]. A resistência é a mesma do BFT: $N \geq 3f+1$.

O protocolo de difusão atômica do Rampart é semelhante ao do BFT. No Rampart, o papel do primário é desempenhado pelo *sequenciador*, que pode ser definido em cada vista, por exemplo, como o servidor com menor identificador. O Rampart tem muito pior desempenho do que o BFT pois usa mensagens assinadas com criptografia de chave pública, em vez de MACs. No entanto, a maior fraqueza do Rampart, que aliás é partilhada com os demais sistemas de comunicação em grupo, é a de que as suspeitas de falha levam à remoção dos suspeitos do grupo. Assim, um atacante pode tentar obter uma maioria de processos maliciosos no grupo atrasando os processos corretos e causando a sua expulsão. Pelo contrário, no BFT se há suspeitas sobre o primário, este passa a secundário, mas nunca é expulso, logo o problema não existe.

Um ponto interessante do Rampart é que pode votar os outputs das réplicas de duas formas diferentes [Reiter, 1995]. A primeira usa um esquema de *assinatura de limiar-(k,N)*: o cliente aceita a resposta se esta estiver assinada usando esse esquema. No entanto,

este esquema apresentou um desempenho fraco de forma a que na prática é usado um esquema de votação semelhante ao BFT.

3.3.1.3. SINTRA, SecureRing, SecureGroup, Worm-IT

Antes de vermos alguns sistemas que melhoram a resistência do BFT e do Rampart, vamos referir quatro sistemas: SINTRA, SecureRing, SecureGroup e Worm-IT. Os quatro têm em comum oferecerem uma primitiva de difusão atômica TI. Não constituem uma solução completa para fazer RME TI, mas o que lhes falta é apenas a comunicação com os clientes. Todos têm resistência $N \geq 3f + 1$.

O sistema SINTRA oferece um conjunto de primitivas de difusão (fiável, atômica, causal), considerando um conjunto estático de máquinas, como o BFT [Cachin and Poritz, 2002]. Ao contrário do BFT e do Rampart que usam detecção de falhas/intrusões para garantir o progresso do sistema, o SINTRA contorna o FLP mediante um protocolo de consenso binário baseado em aleatoridade [Cachin et al., 2000]. O protocolo usa criptografia de limiar e criptografia de chave pública, logo o seu desempenho é fraco quando o tempo de comunicação é "pequeno", como numa LAN; numa WAN o tempo de processamento pode ser negligenciável face ao de comunicação, logo o desempenho pode ser aceitável.

O SecureRing é um sistema de comunicação em grupo especialmente vocacionado para redes de pequena dimensão, já que é baseado num anel lógico de máquinas [Kihlstrom et al., 2001]. O protocolo de ordenação é baseado num *token* que circula no anel, só podendo enviar mensagens quem tiver esse *token*. O sistema usa assinaturas baseadas em criptografia de chave pública, mas em menor quantidade do que o Rampart. Por exemplo, em vez das mensagens serem todas assinadas, é assinado o *token* que transporta um resumo criptográfico das mensagens já enviadas mas ainda não entregues. Máquinas maliciosas são removidas com base em informação fornecida por um detector de falhas bizantinas [Kihlstrom et al., 2003].

O SecureGroup usa um protocolo de difusão atômica baseado em aleatoridade [Moser et al., 2000, Moser and Melliar-Smith, 1999]. Sendo um sistema de comunicação em grupo como o SecureRing, tem o inconveniente de a resistência ser menos de um terço de todas as máquinas do universo das que podem teoricamente entrar no grupo, não das que fazem parte do grupo num determinado instante. Assim, na prática a resistência é menor do que a dos outros sistemas.

O Worm-IT é um sistema de comunicação em grupo baseado na noção de *wormhole* seguro [Correia et al., 2005b, Veríssimo, 2003]. O sistema é baseado num *modelo de falhas híbrido*: a maior parte do sistema pode sofrer intrusões mas cada nó é estendido com um componente seguro, a TTCB [Correia et al., 2002b]. Sobre este tipo de componentes e modelos veremos mais na próxima seção. O que é relevante neste ponto é referir que o protocolo de difusão atômica tem a vantagem de ser eficiente (não usa criptografia de chave pública) e totalmente distribuído (não tem um primário/sequenciador), o que lhe permite evitar os ataques referidos atrás a propósito do BFT.

3.3.1.4. Otimização da resistência

Todos os sistemas mencionados até agora têm um ponto em comum: toleram menos de $1/3$ de réplicas maliciosas, ou seja, $N \geq 3f+1$. Essa proporção pode parecer tão boa como outra qualquer mas, se fizermos as contas, significa que para tolerar intrusões num servidor são precisos $N = 4$, para tolerar intrusões em 2 são precisos $N = 7$ e assim por diante. Estes números são consideravelmente altos, já que cada servidor tem um custo em termos de hardware e software. Mais, para garantir que as réplicas não têm vulnerabilidades, um problema de que falaremos mais tarde, pode ser necessário desenvolver software específico para cada réplica, o que implica um custo significativo. Por tudo isto, reduzir o número de réplicas necessárias para tolerar f intrusões não é uma questão menor.

Uma contribuição engenhosa para esta questão foi proposta em [Yin et al., 2003]. A idéia consiste em separar o *acordo sobre a ordenação de mensagens* da *execução do serviço*. Porque é que todos os sistemas acima precisavam de pelo menos $3f+1$ réplicas? Porque em sistemas assíncronos a difusão atômica – o acordo sobre a ordenação das mensagens – é impossível com menos réplicas. Uma vez ordenados os pedidos/comandos, quantas réplicas são necessárias para executar o serviço? Bastam $2f+1$ para se fazer uma votação simples (conta o que disser a maioria, i.e., $f+1$).

O esquema apresentado nesse artigo é arquitetural: o serviço passa a ter $3f+1$ réplicas que fazem *acordo* sobre a ordenação e $2f+1$ réplicas que *executam* o serviço. Esta arquitetura é apresentada na figura 3.6. A grande vantagem desta solução é que reduz o número de réplicas que executam o serviço que, em geral, serão mais complexas e caras do que as que fazem acordo (a execução do serviço pode envolver por exemplo um base de dados de grande dimensão). É também dada uma solução para proteger a confidencialidade dos dados nos servidores através de uma *firewall*, mas isso ficará para mais tarde (seção 3.6).

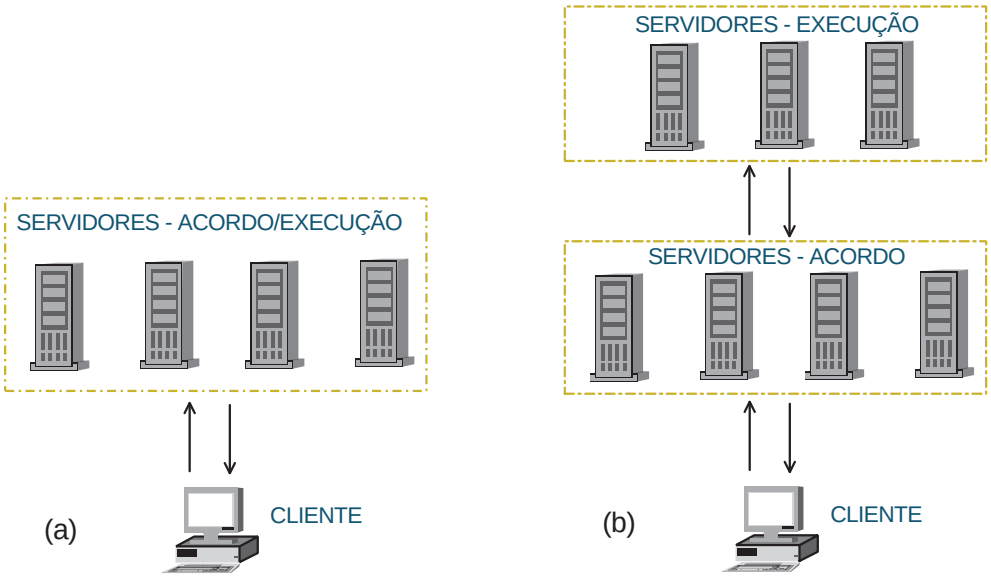

Figura 3.6. Separação acordo-execução. (a) Arquitetura típica. (b) Arquitetura com separação [Yin et al., 2003].

Um sistema que permite realmente diminuir o número de réplicas para $N \geq 2f+1$ é apresentado em [Correia et al., 2004]. A solução consiste em usar um *modelo de falhas híbrido* baseado num *wormhole* seguro para ordenar as mensagens[12]. Este *wormhole*, chamado TTCB, é um componente distribuído e seguro. A arquitetura do sistema está representada na figura 3.7. A TTCB é um núcleo de segurança distribuído, que é suficientemente simples para ser construída de forma a que seja segura, ou seja, para que não seja possível que aí ocorram intrusões. Na realidade, para suportar este serviço o componente precisa de ter apenas um serviço de ordenação de mensagens: o TMO (*Trusted Multicast Ordering*). Como esse serviço é executado num componente seguro, são precisos apenas $2f+1$ servidores para o executar.

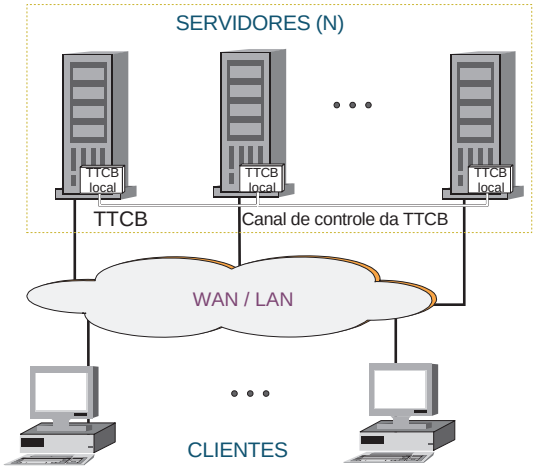

Figura 3.7. Replicação de máquinas de estados com um *wormhole* [Correia et al., 2004].

O algoritmo funciona esquematicamente da seguinte forma. Um cliente envia um pedido para um dos servidores à sua escolha. Um servidor malicioso pode tentar não processar o pedido, logo ao fim de um certo tempo o cliente reenvia o pedido para outros f servidores, o que garante que pelo menos um servidor correto o recebe. O pedido vai assinado com um vetor de MACs de forma a evitar que um servidor malicioso modifique o pedido.

Quando um servidor correto recebe o pedido, difunde-o por todos os outros servidores e passa um resumo criptográfico desse mesmo pedido ao TMO. Quando outro servidor recebe esse pedido, passa também uma síntese ao TMO. Quando o TMO descobre que $f+1$ servidores têm o mesmo pedido, atribui-lhe um número de ordem e entrega esse número a todos os servidores. O TMO espera por $f+1$ servidores para garantir que pelo menos um é correto e tem mesmo a mensagem. Como o TMO é seguro, não pode "mentir", todos os servidores recebem o mesmo número de ordem para o pedido. Quando um servidor não tem pedidos com número menor por executar, executa o comando nesse pedido e envia a resposta ao cliente. Quando o cliente recebe $f+1$ cópias da mesma res-

[12] A metáfora do *wormhole* vem de um conceito da astrofísica que alguma fição científica tem apresentado como atalhos que permitiram viajar em pouco tempo entre pontos afastados do universo. Uma introdução ao tema está em http://en.wikipedia.org/wiki/Wormhole. A idéia explorada neste trabalho consiste em usar um componente seguro por onde a informação pode "viajar" em segurança. Os *wormholes* têm sido usados também para aplicações com requisitos temporais [Veríssimo and Casimiro, 2002].

posta vindas de servidores diferentes tem a certeza de que essa é a resposta correta pois pelo menos um dos servidores é correto. Logo, aceita essa resposta.

O algoritmo tem muito mais detalhes que não podem ser aqui explicados. Convém apenas dizer que não usa criptografia de chave pública em tempo de execução (pode ser necessária para distribuir inicialmente as chaves secretas).

3.3.2. Quoruns

Um *sistema de quoruns* $\mathscr{Q}$ é um conjunto de subconjuntos de servidores denominados *quoruns* tal que $\forall Q_1, Q_2 \in \mathscr{Q}, Q_1 \cap Q_2 \neq \emptyset$. Esta definição, apesar de comum, tem tanto de precisa quanto de obscura. Para que servirá definir *um conjunto de subconjuntos de servidores*?

Dado um conjunto de servidores U, um sistema de quoruns permite raciocinar sobre esses servidores e definir *objetos* distribuídos com diferentes semânticas, por exemplo, variáveis compartilhadas, objetos de exclusão mútua e objetos de consenso. Esta explicação não exclui os algoritmos de RME, que no fundo concretizam um tipo de variáveis compartilhadas com semântica forte (leituras e escritas ordenadas). A diferença é que enquanto que a RME é uma solução genérica para concretizar *serviços* tolerantes a faltas/intrusões, os quoruns geralmente são usados para construir *repositórios de dados* tolerantes a faltas/intrusões, o que constitui um caso particular dos referidos serviços.

Ao servirem para concretizar algo de mais simples do que RME, muitas vezes os trabalhos com quoruns evitam a necessidade de realizar consenso e, como tal, não são circunscritos pelo FLP podendo os algoritmos ser totalmente assíncronos. No entanto, a principal diferença entre a RME e os sistemas de quoruns é que as operações na RME envolvem sempre todos os servidores, enquanto que nos sistemas de quoruns as operações são geralmente feitas sobre um quorum – um subconjunto dos servidores – o que torna os algoritmos mais escaláveis.

Nesta seção vamos considerar o uso de sistemas de quoruns para concretizar repositórios de dados. Uma forma de caracterizar um repositório é pensando nele como concretizando um conjunto de *variáveis de memória compartilhada* (*shared-memory*), ou seja, de memória que pode ser lida e escrita por diversos processos (clientes). Em sistemas distribuídos há um trabalho vasto em *algoritmos sobre memória compartilhada*, desenvolvido em paralelo com todo o trabalho em *algoritmos com comunicação por mensagens* (*message passing*)[13]. No entanto, apesar de serem duas linhas de pesquisa paralelas, num sistema distribuído as variáveis de memória compartilhada têm necessariamente de ser concretizadas usando clientes e servidores que se comunicam por mensagens. Os sistemas de quoruns fornecem uma forma conveniente de raciocinar sobre os conjuntos de servidores com vista à definição de algoritmos que implementem os repositórios.

Lamport apresentou uma classificação de variáveis de memória compartilhada que continua a ser amplamente utilizada [Lamport, 1986]. Um primeiro ponto dessa classificação é o de quantos processos podem acessar uma variável para leitura e escrita. Na discussão que se segue vamos considerar sempre variáveis com múltiplos-escritores/múltiplos-leitores (*multi-writer/multi-reader*), embora existam muitos traba-

[13]Os algoritmos mencionados neste capítulo são todos baseados em comunicação por mensagens.

lhos em variáveis para um-escritor/múltiplos-leitores (*single-writer/multi-reader*).

O segundo ponto da classificação é a *semântica de consistência* da variável, que pode ser uma de três: segura (*safe*), regular e atômica. Diz-se que a operação o_1 *acontece antes* da operação o_2 se o_1 termina antes de o_2 começar. Duas operações o_1 e o_2 dizem-se *concorrentes* se nem o_1 *acontece antes* de o_2 nem o_2 *acontece antes* de o_1. As três semânticas de consistência podem ser definidas informalmente da seguinte forma [Lamport, 1986, Martin et al., 2002a]:

- *segura:* uma leitura que não seja concorrente com nenhuma escrita retorna o último valor escrito; uma leitura concorrente com uma ou mais escritas retorna qualquer valor;
- *regular:* garante a semântica segura e também que se uma leitura é concorrente com várias escritas, o valor retornado é um dos valores dessas operações de escrita ou o valor escrito pela última escrita que tenha terminado antes da leitura;
- *atômica:* garante a semântica regular e também que as escritas e leituras retornam valores como se tivessem sido feitas de acordo com uma ordem definida; uma variável com esta semântica também se diz *linearizável* [Herlihy and Wing, 1990][14].

Estas definições seguem as originais de [Lamport, 1986] que não consideram o caso de múltiplos escritores, apenas múltiplos leitores. Alguns trabalhos mais recentes, como [Martin et al., 2002a], consideram também este caso. A especificação da semântica no caso de escritas concorrentes depende de cada algoritmo. É interessante notar que se fosse usada replicação de máquinas de estados para concretizar um repositório de dados, p. ex. o BFT, a semântica obtida seria a mais forte, a atômica, incluindo ordem nas escritas concorrentes.

Os trabalhos em quoruns para tolerância a faltas acidentais tem vários anos e é até anterior à replicação de máquinas de estados [Gifford, 1979]. Muito mais recentemente surgiu um interesse considerável no estudo de sistemas de quoruns tolerantes a faltas bizantinas, ou tolerantes a intrusões, começando em [Malkhi and Reiter, 1997a, Malkhi et al., 1997][15]. Estes sistemas são chamados de *sistemas de quoruns de mascaramento*, já que o objetivo é *mascarar* a ocorrência de faltas em alguns servidores.

3.3.2.1. Variáveis compartilhadas com quoruns

O tipo de objeto distribuído mais óbvio é a *variável compartilhada*. O trabalho em sistemas de quoruns é fértil em algoritmos para a concretização deste tipo de variáveis. Vamos ilustrar este tipo de algoritmos com variantes das variáveis compartilhadas definidas em [Malkhi and Reiter, 1998a, Malkhi and Reiter, 1998b][16].

[14]Na realidade o conceito de linearizável é uma generalização da semântica atômica de variáveis compartilhadas para objetos genéricos.

[15]Há um trabalho anterior que combina quoruns e faltas bizantinas mas que considera apenas clientes maliciosos [Naor and Wool, 1996].

[16]O trabalho original usa a noção de *fail-prone system* para generalizar a idéia de que *no máximo f servidores podem falhar* [Malkhi and Reiter, 1998a]. Na prática tal generalização conduz a algoritmos

Para cada variável compartilhada x existe em cada servidor u uma cópia da variável x_u e uma estampilha temporal $t_{x,u}$. No sistema existe um conjunto de escritores W_x que contém os identificadores dos clientes que podem escrever em x. O valor de W_x é conhecido por todos os clientes e servidores (pode estar codificado no nome da variável ou ser guardado em outra variável compartilhada). A estampilha temporal indica quando a variável foi escrita pela última vez e os conjuntos de estampilhas temporais usadas pelos clientes não se intersectam (p. ex. as próprias estampilhas incluem o identificador do cliente nos bits menos significativos).

Os clientes comunicam com os servidores usando uma *chamada a procedimento remoto quorum*. A chamada Q-RPC(m) envia o pedido m a um subconjunto dos servidores e recolhe respostas de um quorum. A operação pode implicar reenviar o pedido e excluir servidores maliciosos, já que estes podem tentar boicotar o funcionamento do sistema de diferentes modos. Nem os clientes e nem os servidores comunicam diretamente entre si, o que é uma característica comum à maioria dos trabalhos de quoruns publicados, que traz benefícios em termos de escalabilidade [Malkhi and Reiter, 2000].

Os sistemas de quoruns podem ser divididos em várias classes, das quais veremos duas. Começaremos pela classe que conduz a algoritmos mais simples, os sistemas de quoruns de *disseminação-f*. A principal característica destes sistemas é que os dados armazenados são assinados pelo cliente que os escreve – *dados auto-verificáveis* – logo os servidores não podem forjar ou modificar esses dados. Nos algoritmos de escrita e leitura que vamos ver, o sistema de quoruns tem de obedecer a duas propriedades:

- *Consistência.* $\forall Q_1, Q_2 \in \mathscr{Q}, |Q_1 \cap Q_2| \geq f+1$
- *Disponibilidade.* $\forall Q \in \mathscr{Q}, |Q| \leq n-f$

Comecemos por assumir que *os escritores são corretos*[17], que $N \geq 3f+1$ (consequência das duas propriedades acima) e que o tamanho dos quoruns é $\forall Q \in \mathscr{Q}, |Q| = \lceil \frac{N+f+1}{2} \rceil$. O algoritmo que permite escrever o valor v na variável compartilhada x com *semântica regular* é bastante simples:

1. fazer Q-RPC para obter o conjunto de estampilhas temporais $\{t_{x,u}\}_{u \in Q_1}$ do quorum Q_1 de servidores;
2. escolher uma estampilha t maior do que todas as obtidas e fazer Q-RPC para enviar o par $\langle v,t \rangle$ para um quorum Q_2.

É fácil compreender que f servidores maliciosos não podem interferir com o funcionamento do algoritmo pois os valores que não estiverem corretamente assinados por um escritor de W_x são descartados pelos leitores que os recebam, e assumiu-se que os escritores são corretos, logo os valores serão assinados corretamente. O algoritmo que lê o conteúdo de x é igualmente simples:

---

mais difíceis de compreender, logo não vamos usá-la.

[17] Podíamos falar de *clientes* corretos, mas sob o ponto de vista do algoritmo é irrelevante os leitores serem corretos ou maliciosos já que não alteram o estado dos servidores.

1. fazer Q-RPC para obter o conjunto de pares assinados valor/estampilha $\{\langle v_u, t_u\rangle_{w_u}\}_{u\in Q_1}$ no quorum Q_1 de servidores;

2. retornar o valor com a maior estampilha escrito por um escritor de W_x.

Consideremos agora a possibilidade de existirem *escritores maliciosos*. O problema é complicado pois um escritor malicioso pode facilmente deixar o sistema num estado inconsistente, p. ex. escrevendo um par $\langle v_i, t_i\rangle$ diferente em cada servidor. A primeira parte da solução consiste em usar o protocolo *echo* [Reiter, 1994] para garantir que todos os servidores que aceitem a escrita de um valor, aceitem o mesmo par $\langle v,t\rangle$. O protocolo funciona assim: (1) o escritor envia o par para os servidores e obtém ecos assinados vindos de um quorum inteiro; (2) o escritor envia o par e as assinaturas para os servidores do mesmo quorum. Como todos os quoruns se intersectam é impossível o servidor escrever em dois quoruns dois valores diferentes com a mesma estampilha, ou seja, dois pares $\langle v,t\rangle$ e $\langle v',t\rangle$ com $v \neq v'$.

Consideremos agora o caso genérico em que não temos dados assinados pelos escritores, ou seja, em que os dados não são *auto-verificáveis*. Uma justificativa para esta generalização é a de que para verificar as assinaturas seria preciso distribuir as chaves públicas dos clientes pelos outros clientes, o que pode não ser conveniente.

Neste caso genérico é necessário que a interseção de dois quoruns tenha sempre uma maioria de servidores corretos:

- *Consistência.* $\forall Q_1, Q_2 \in \mathscr{Q}, |Q_1 \cap Q_2| \geq 2f+1$

Assim, é necessário aumentar a redundância de servidores: $N \geq 4f+1$ e $\forall Q \in \mathscr{Q}, |Q| = \lceil \frac{N+2f+1}{2} \rceil$. Este tipo de sistema de quoruns é denominado de *mascaramento-f*.

Um algoritmo que permite a um escritor $w \in W_x$ escrever o valor v na variável compartilhada x com *semântica segura* é:

1. fazer Q-RPC para obter o conjunto de estampilhas temporais $\{t_{x,u}\}_{u\in Q_1}$ do quorum Q_1 de servidores;

2. escolher uma estampilha t maior do que todas as obtidas;

3. fazer Q-RPC para enviar $\langle v,t\rangle$ e receber ecos assinados de um quorum Q_2;

4. fazer Q-RPC para enviar os ecos para Q_2, escrevendo $\langle v,t\rangle$ nesse quorum.

O protocolo que permite a um cliente ler a variável x é:

1. fazer Q-RPC para obter o conjunto de pares valor/estampilha $\{\langle v_u, t_{x,u}\rangle_{w_u}\}_{u\in Q_1}$ no quorum Q_1 de servidores, cada par assinado pelo servidor onde se encontra;

2. retornar o valor v do par $\langle v,t\rangle$ com maior estampilha que apareça em pelo menos $f+1$ respostas (ou $\perp$ se não houver nenhum).

Caso uma leitura não seja concorrente com uma ou mais escritas, o valor retornado é o último escrito. Caso contrário, a variável pode retornar um valor qualquer dos que estão a ser escritos ou $\perp$. Essa é a razão pela qual a semântica é a mais fraca, a semântica *segura*.

Os algoritmos apresentados têm em comum a forma como os quoruns aí aparecem explicitamente (p. ex. Q_1, Q_2). Esta forma de especificação torna os algoritmos muito genéricos já que não dependem do conteúdo dos quoruns. Uma forma menos genérica mas mais simples de especificar quoruns é a que referimos atrás a propósito do BFT, por exemplo, "um quorum de (quaisquer) $f+1$ servidores", "um quorum de $2f+1$ servidores" [Castro and Liskov, 2002, Martin et al., 2002a, Martin and Alvisi, 2004].

Em relação às variáveis compartilhadas vale a pena dizer ainda que alguns algoritmos mais eficientes dos que aqui apresentados encontram-se em [Martin et al., 2002a]. A tabela 3.1 apresenta uma comparação de alguns algoritmos. Um protocolo *não-confirmável* difere dos que vimos em que não é possível determinar quando uma escrita termina.

referência	disseminação-f confirmável	mascaramento-f confirmável	disseminação-f não-confirmável	mascaramento-f não-confirmável
[Malkhi and Reiter, 1998a]	regular, 3f+1	segura, 4f+1		
[Malkhi and Reiter, 1998b]	atômica, 3f+1	segura, 4f+1		
[Martin et al., 2002a]	atômica, 3f+1	atômica, 3f+1	regular, 2f+1	regular, 2f+1
[Martin et al., 2002b]	regular, 3f+1	segura, 4f+1	regular, 2f+1	segura, 3f+1

Tabela 3.1. Comparação das semânticas e resistências de variáveis compartilhadas com quoruns [Martin et al., 2002a].

3.3.2.2. Outros objetos de memória compartilhada com quoruns

Vários outros objetos de memória compartilhada têm sido concretizados usando quoruns. Uma primitiva importante quando se fala de programação concorrente é a *exclusão mútua*. No tipo de sistemas que estamos a ver, o objetivo é permitir a um cliente reservar recursos (p. ex. objetos) para seu uso exclusivo enquanto realiza determinada operação. Um algoritmo que não garante que o recurso seja reservado caso haja vários clientes a concorrer é apresentado em [Malkhi and Reiter, 1998b]. Um algoritmo de exclusão mútua sem esta restrição baseado no algoritmo do confeiteiro de Lamport encontra-se em [Bessani et al., 2005]. Apesar de os algoritmos com quoruns tentarem geralmente evitar o consenso, por vezes é mesmo necessário ter um objeto que realize essa operação. Um objeto de consenso aleatório baseado em quoruns encontra-se em [Malkhi and Reiter, 2000].

Por último, vale a pena referir um trabalho recente que modifica uma premissa que todos os sistemas de quoruns até então assumiam: f e N serem constantes. Martin e Alvisi introduziram uma metodologia genérica que permite transformar protocolos de quoruns como os dados acima em protocolos dinâmicos, nos quais f e N podem crescer ou diminuir [Martin and Alvisi, 2004]. A idéia básica consiste em substituir a primitiva Q-RPC por uma primitiva DQ-RPC que lida com as variações desses parâmetros.

3.4. Fragmentação: garantindo disponibilidade, integridade e confidencialidade

Voltemos à nossa estória do pirata. As soluções de replicação que foram apresentadas resolvem o problema da disponibilidade e da integridade de um serviço se um máximo de f torreões forem invadidos pelo pirata. Imaginemos agora que o que o pirata procura é informação, por exemplo, um mapa do tesouro. Nesse caso, replicar informação em todos os torreões, em lugar de aumentar a segurança acaba por diminuí-la, já que basta ao pirata tomar um dos torreões para o obter o mapa! Para garantir também a *confidencialidade* – além da disponibilidade e integridade – os dados terão de ser *fragmentados* ou *disseminados* pelos diversos servidores/torreões, podendo ser reconstruídos apenas por quem tiver autorização para o fazer. Se o pirata invadir um torreão obterá apenas um pedaço de mapa sem significado.

A fragmentação de dados para obter disponibilidade, integridade e confidencialidade é o tema desta seção. Convém referir que alguns trabalhos que veremos têm outra motivação: fragmentar os dados pelos servidores para diminuir o espaço de armazenamento total, evitando ter uma réplica de todos os dados em todos os servidores. Em qualquer dos casos, nesta linha de trabalho o objetivo é o *armazenamento de dados*, como vimos a propósito dos sistemas de quoruns, não a concretização de serviços genéricos, como na replicação de máquinas de estados.

O trabalho seminal na área é de Fraga e Powell, que cunharam originalmente o termo *tolerância a intrusões* [Fraga and Powell, 1985]. Esse trabalho tratou de muitos dos problemas que os trabalhos seguintes também trataram, logo é por ele que vamos começar.

Esse trabalho apresenta um sistema de arquivos TI e introduz uma técnica chamada *fragmentação-redundância-dispersão* (*FRS, fragmentation-redundancy-scattering*). A idéia básica é fácil de compreender. Um sistema de arquivos é concretizado através de um conjunto de servidores. Cada arquivo F, antes de ser armazenado, é fragmentado em m fragmentos, que depois são espalhados pelos N servidores, de tal modo que cada fragmento fique guardado em vários servidores (para garantir a disponibilidade) e nenhum servidor tenha fragmentos suficientes para que um intruso possa reconstruir F (para contribuir para a confidencialidade) – v. fig. 3.8[18]. Desta breve explicação percebe-se imediatamente que o sistema não fornece propriamente confidencialidade em sentido estrito, já que um intruso pode obter alguma informação atacando um servidor, embora para reconstruir o arquivo completo sejam necessários m fragmentos. Para usar as palavras do artigo, este esquema "reduz o significado da informação disponível a um intruso". Em relação à integridade, são sugeridas duas soluções alternativas para detectar fragmentos corrompidos: (1) quando é feita a leitura são lidas diversas cópias de cada fragmento e faz-se uma votação; (2) junta-se um MAC a cada fragmento.

A informação sobre a localização dos arquivos está armazenada num serviço que é também distribuído para tolerar intrusões. Um ponto importante gerido por esse serviço

[18]Nos trabalhos desta área é usual falar-se de fragmentação, ou dispersão, de um arquivo F, em vez de acesso a uma variável compartilhada x, como nos trabalhos sobre quoruns. Nesta seção vamos seguir essa nomenclatura.

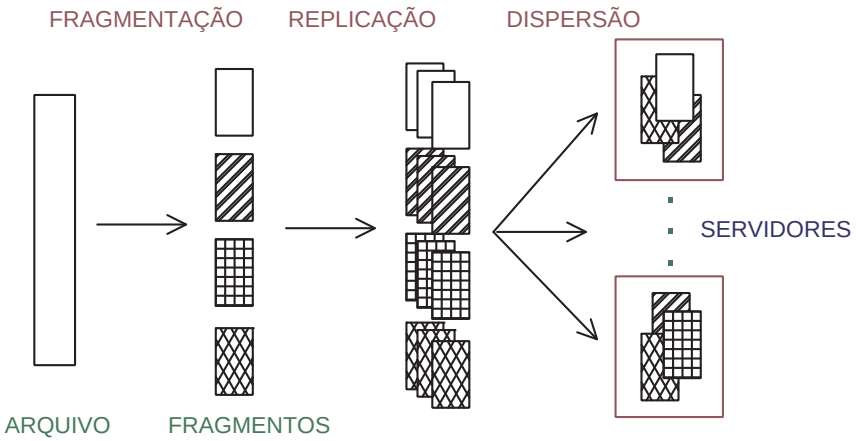

Figura 3.8. Fragmentação-redundância-dispersão [Deswarte et al., 1991].

é o da autorização de acesso a um arquivo. A solução, que é apenas esboçada, usa um esquema de partilha de segredos para construir a chave necessária para acessar ao arquivo. O sistema tem a limitação de não tolerar clientes maliciosos.

3.4.1. Códigos de apagamento

Pouco depois de [Fraga and Powell, 1985], Rabin publicou uma solução para fragmentação – a que chama um *algoritmo de dispersão de informação* – que otimiza o espaço ocupado com base em *códigos de apagamento* (*erasure codes*) [Rabin, 1989]. Estes códigos são semelhantes aos códigos de correção de erros usados em telecomunicações, mas enquanto nos primeiros a informação pode apenas ser apagada, nos segundos pode também ser modificada.

A idéia consiste em dividir um arquivo em N fragmentos de forma a que seja suficiente ter k fragmentos para reconstruí-lo, mas $k-1$ fragmentos não cheguem para o fazer. Para o efeito usa-se um *código de apagamento-(k,N)*. Esse trabalho não fornece propriamente um protocolo para concretizar o esquema num sistema distribuído. Esse passo foi dado em outros trabalhos que dele derivam [Krawczyk, 1993, Alon et al., 2000, Garay et al., 2000]. Mais tarde, no âmbito do projeto PASIS (programa OASIS) foi definido um sistema eficiente para dispersão de informação com semântica atômica, que tolera clientes maliciosos [Goodson et al., 2004]. Nenhum destes trabalhos aborda a questão da confidencialidade, sendo os códigos de apagamento usados unicamente para otimizar o espaço ocupado pelos dados.

O único trabalho dessa linha que considera o problema da confidencialidade dos dados é bastante recente [Cachin and Tessaro, 2004, Cachin and Tessaro, 2005]. O primeiro dos dois artigos não permite acessos concorrentes, e cada arquivo só pode ser escrito uma vez (não lida com versões). Mesmo assim, esse trabalho vai servir para ilustrar este tipo de soluções.

O primeiro mecanismo apresentado em [Cachin and Tessaro, 2004] é denominado AVID (*asynchronous verifiable information dispersal*) e não fornece confidencialidade, apenas integridade e disponibilidade. Os clientes podem ser maliciosos.

Um cliente que quer armazenar um arquivo F começa por o codificar como um vetor $[F_1,...,F_n]$ usando um código de apagamento-(k,N). Além disso obtém um conjunto de *impressões digitais* [Krawczyk, 1993] calculando um vetor com sínteses criptográficas (*hashes*) de cada F_i: $D = [D_1,...,D_n]$. Depois, toda essa informação é enviada para os servidores usando um protocolo de difusão fiável, que é uma variante do protocolo clássico de Bracha [Bracha, 1984]. Um protocolo de difusão fiável garante duas propriedades: (1) todos os servidores entregam os mesmos pedidos; (2) se o cliente é correto, o pedido é entregue (se for malicioso, pode não ser entregue). Este protocolo não é usado *ipsis verbis* mas modificado para diminuir a quantidade de dados enviados: o cliente envia apenas F_i para o servidor s_i, mas depois estes servidores trocam entre si os F_i para garantirem que todos têm o seu fragmento. Se o cliente for malicioso e alguns dos fragmentos estiverem corrompidos há duas possibilidades: o número de fragmentos disponíveis permite reconstruir os fragmentos omitidos, o que é feito; ou não é possível reconstruir esses fragmentos e o arquivo não é armazenado. Quando a operação é terminada os servidores apagam todos os fragmentos que não lhe pertencem.

A operação de leitura consiste simplesmente em pedir fragmentos aos servidores até se obterem os k necessários para reconstruir F. O parâmetro k tem de verificar a condição: $f+1 \leq k \leq N-2f$. A melhor resistência é obtida quando $N = 3f+1$, logo $k = f+1$.

O mesmo artigo apresenta o esquema cAVID que garante também a confidencialidade dos dados armazenados. Para garantir a confidencialidade é necessário haver controle de acesso ao arquivo. Para o efeito junto do arquivo é guardada uma lista de controle de acesso L com os identificadores dos clientes que a ele podem acessar.

A forma como é conseguida a confidencialidade é simples: o arquivo é cifrado usando criptografia simétrica antes de ser armazenado usando o esquema AVID. O problema é o que se faz da chave. Se o cliente ficasse com a chave para si, só ele poderia recuperar o arquivo, o que em geral não é o objetivo. Para resolver este problema usa-se um esquema de *criptografia de limiar*. Este esquema fornece essencialmente:

- um algoritmo para cifrar dados usando uma *chave pública PK*;
- um algoritmo para decifrar dados que usa uma *chave privada* SK_i para obter um fragmento σ dos dados cifrados;
- um algoritmo que permite verificar se um fragmento σ é válido usando uma *chave de verificação* VK;
- um algoritmo que permite obter os dados iniciais combinando k fragmentos decifrados usando VK.

Cada servidor tem uma chave privada SK_i e todos os clientes têm a chave pública PK. Para armazenar o arquivo F, o cliente gera uma chave secreta K, cifra o arquivo com essa chave (usando criptografia simétrica), e cifra K com PK. Depois, usa o algoritmo de dispersão AVID para armazenar o arquivo, a chave K cifrada e a lista de controle de acesso L. Para ler o arquivo, é necessário obter fragmentos σ de k servidores para reconstruir K.

Note que este esquema é ortogonal à fragmentação, podendo ser adicionado a um serviço baseado em replicação de máquinas de estados ou quoruns para garantir a confidencialidade dos dados armazenados. No entanto, se o serviço for baseado em fragmentação, após o processo de armazenamento do arquivo é necessário penetrar em k servidores para poder executar um ataque com o objetivo de quebrar a cifra de K, enquanto que com RME ou quoruns basta penetrar em um, logo a segurança oferecida é ligeiramente superior.

Este esquema é estendido para concretizar uma variável compartilhada para múltiplos escritores e múltiplos leitores e com semântica atômica em [Cachin and Tessaro, 2005]. A solução é baseada em estampilhas temporais de forma semelhante ao usado em sistemas de quoruns. Por exemplo, para fazer uma escrita o cliente começa por obter a maior estampilha armazenada e depois escreve o arquivo com uma estampilha superior. Para evitar ataques de negação de serviço através da utilização de estampilhas muito grandes, é usado o esquema de *non-skipping timestamps* que não permite que estas tomem um valor superior ao número de escritas já realizadas [Bazzi and Ding, 2004].

3.4.2. Partilha de segredos

Uma solução bastante evidente para garantir a confidencialidade dos dados seria usar um esquema de *partilha de segredos*, como os de Shamir ou Blakley [Shamir, 1979, Blakley, 1979]. No entanto, só foi encontrado um trabalho que usa essa técnica para tolerar intrusões em servidores, a *secure store* [Lakshmanan et al., 2003].

A razão pela qual não surgiram mais sistemas deste tipo é fácil de conjeturar: dificilmente o desempenho de um esquema de partilha de segredos é compatível com um sistema de armazenamento de dados de tamanho arbitrário. Para resolver esse problema, o trabalho que vamos estudar combina essa técnica com replicação, necessitando assim de mais servidores do que os sistemas que vimos até agora. Os servidores formam uma matriz com c colunas e r linhas, num total de $N = rc$ servidores (v. fig. 3.9).

Quando um cliente pretende armazenar um arquivo, divide-o em c fragmentos usando um *mecanismo de partilha de segredos (k,c)*, sendo k o número de fragmentos necessário para reconstruir o segredo (os mínimos são $k = f + 1$ e $c = 2f + 1$). Depois, cada fragmento é replicado por uma coluna de servidores (v. figura). Para recuperar um arquivo é preciso obter k fragmentos de diferentes colunas.

O sistema oferece um mecanismo ingênuo de recuperação proativa, o tipo de mecanismo que veremos na próxima seção. Em vez de simplesmente assumir que o número máximo de intrusões é f, o sistema considera que esse é o número máximo de intrusões durante um intervalo de tempo T_v. Para garantir essa hipótese, os fragmentos são renovados cada $T < T_v$, ou seja, são novamente cifrados usando um *protocolo de renovação de fragmentos*. Obviamente isso não é particularmente útil pois no mínimo é também necessário remover a intrusão ou o número de servidores corrompidos irá sempre aumentando.

A *secure store* não considera clientes maliciosos, que podem deixar os servidores num estado inconsistente. As semânticas de consistência oferecidas são fracas.

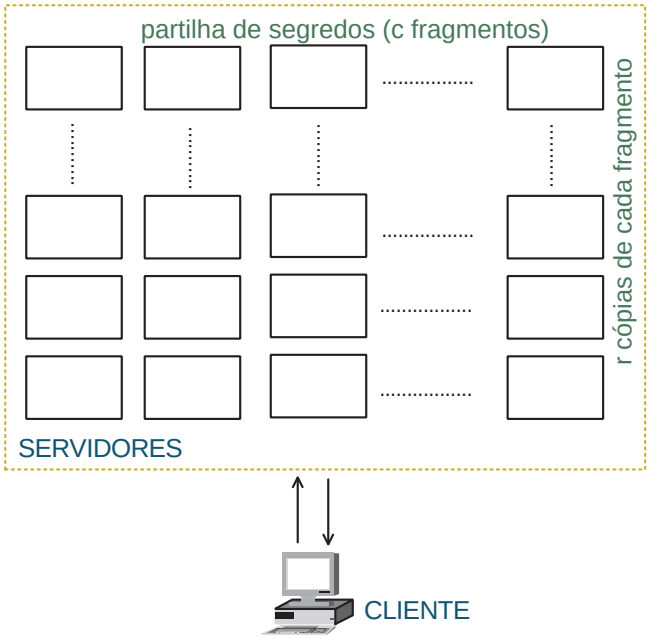

Figura 3.9. Arquitetura da *secure store* [Lakshmanan et al., 2003].

Um comentário final sobre a relação de todas estas soluções para TI com confidencialidade. Alguns dos artigos discutidos falam de *quoruns*. De fato, faz todo o sentido falar de sistemas de quoruns também neste contexto, embora com a diferença de que agora os dados não são replicados pelos servidores mas fragmentados. Na realidade, a utilização de sistemas de quoruns é ortogonal a todos os esquemas que temos visto, embora por vezes surjam apenas implicitamente.

3.5. Recuperação proativa

O mascaramento de intrusões através de replicação (seção 3.3) ou fragmentação (seção 3.4) permite tolerar um número máximo de f intrusões em servidores durante o tempo de vida de um serviço. Esse tempo normalmente será longo, p. ex. de meses, tornando essa premissa de existir um número máximo de intrusões difícil de substanciar. Lembremo-nos do nosso pirata: se lhe for dado tempo suficiente ele pode acabar por invadir muitos torreões!

Para ultrapassar essa limitação é necessário utilizar uma técnica de *tolerância a faltas* que referimos na seção 3.2.1: o *processamento de erros*. No contexto da TI, a idéia consiste em remover as intrusões que vão ocorrendo de tal forma a que o número de servidores corrompidos nunca ultrapasse f. Apesar de alguns trabalhos sugerirem a utilização de *detectores de intrusões* para descobrir quando é necessário remover uma intrusão, os sistemas desse tipo atualmente disponíveis dificilmente poderão ser usados para fazer remoção automática de intrusões dado o número elevado de falsos positivos e falsos negativos que produzem [Lippmann et al., 2000].

Na prática, na TI tem sido usado um mecanismo de processamento de erros denominado *recuperação proativa*. A idéia consiste em fazer periodicamente uma renovação de cada servidor de forma a deixá-lo num estado correto. Em outras palavras,

periodicamente são removidas quaisquer alterações ao estado ou código de um servidor no qual tenha ocorrido uma intrusão; esta renovação é feita mesmo que não tenham acontecido intrusões. Esta técnica não faz o sistema ficar invulnerável, mas o pirata teria de ser muito rápido para conseguir violar a sua segurança. Na prática o risco de serem corrompidos mais do que f servidores fica circunscrito a uma *janela de vulnerabilidade* que depende do período de recuperação.

Um atacante que realiza este tipo de ataques que procura corromper sucessivos servidores é por vezes chamado de *adversário móvel*, seguindo o primeiro trabalho na área [Ostrovsky and Yung, 1991]. Um interessante resumo dos trabalhos mais antigos nesta linha encontra-se em [Canetti et al., 1997]. Os trabalhos que veremos em seguida são mais recentes.

3.5.1. BFT-PR

O sistema BFT (seção 3.3.1.1) foi estendido para fazer recuperação proativa, passando a chamar-se BFT-PR [Castro and Liskov, 2002]. A idéia básica desta recuperação é a que acabamos de ver. As soluções de recuperação proativa anteriores ao BFT-PR, começando em [Ostrovsky and Yung, 1991], exigiam que o código da réplica estivesse em memória só de leitura. O BFT-PR precisa apenas de um pequeno *monitor* em memória deste tipo. No BFT-PR a recuperação consiste em realizar três operações:

1. renovar as chaves secretas usadas para a comunicação cliente-servidor e servidor-servidor (usadas para obter os MACs);
2. repor o código do sistema caso tenha sido corrompido;
3. repor o estado do sistema caso tenha sido corrompido.

O artigo em questão aponta diversas premissas para este tipo de recuperação ser possível. Em cada réplica tem de existir:

- um coprocessador criptográfico que armazene a chave privada da réplica e assine e decifre mensagens sem expor essa chave;
- uma memória não volátil só de leitura onde sejam guardadas as chaves públicas de cada uma das outras réplicas e o monitor de recuperação (p. ex. a BIOS);
- um temporizador seguro para disparar a recuperação (existem temporizadores em hardware que podem ser usados com esse fim).

Todos estes mecanismos exigem a hipótese adicional de que o atacante não tem acesso físico ao servidor. O BFT-PR exige ainda uma hipótese temporal mais forte do que o BFT: existe um instante desconhecido a partir do qual o atraso da comunicação é inferior a um dado Δ (ou seja, um modelo de sincronia parcial [Dwork et al., 1988]).

A primeira operação de recuperação referida acima é a renovação das chaves secretas. O protocolo usado é extremamente simples pois consiste no envio periódico (p. ex. cada minuto) de uma única mensagem com a nova chave. A mensagem enviada

pelo servidor s_i ao servidor s_j tem a forma: $\langle \text{NEW-KEY}, i, ..., \{k_{j,i}\}_{\varepsilon_j}, ..., t \rangle_{\sigma_i}$. A chave $\{k_{j,i}\}$ é a que será usada por s_j para calcular os MACs enviados a s_i. A chave é cifrada com a chave pública de s_j (ε_j) de forma a que só s_j a possa decifrar. A mensagem inclui um contador t que evita ataques de repetição de mensagens antigas. A mensagem é assinada com a chave privada de s_i (σ_i). Estas chaves são usadas para proteger a comunicação numa única direção. Já a comunicação com o cliente é protegida nas duas direções usando uma única chave, que lhe é enviada pelo servidor usando uma mensagem do mesmo tipo.

A segunda operação de recuperação consiste em repor o código do servidor. Esta recuperação é feita de forma proativa sempre que o temporizador dispara. Quando isso acontece, o monitor cria uma imagem do código e do estado da réplica em disco. Depois, força o reinício da máquina (*reboot*). Para verificar se o sistema operacional ou o código do serviço foram corrompidos o monitor usa as suas sínteses criptográficas que se encontram guardadas na memória só de leitura. Se tiverem sido corrompidos, o monitor obtém uma cópia de outros servidores.

A terceira operação começa por executar um protocolo com os outros servidores para determinar se o estado da réplica foi corrompido. Se tiver sido corrompido, o estado é transferido de outras réplicas corretas.

A janela de vulnerabilidade do BFT-PR é $T_v = 2T_k + T_r$, sendo T_k o período máximo de renovação das chaves e T_r o período de recuperação do servidor.

3.5.2. COCA

O sistema COCA (*Cornell On-line Certification Authority*) é uma autoridade de certificação *on-line* TI desenvolvida no âmbito do programa OASIS [Zhou et al., 2002a]. Esse sistema é bem mais simples do que o BFT-PR mas tem o interesse de ser um dos primeiros a usar recuperação proativa e de o fazer de uma forma diferente da que vimos.

O objetivo do COCA é fornecer certificados com associações entre nomes e chaves públicas. O sistema oferece apenas duas operações:

- *Update:* criar, atualizar ou invalidar certificados.
- *Query:* obter o certificado correspondente a um nome.

O COCA não usa RME mas um sistema de quoruns de disseminação. O número de servidores é de $N \geq 3f + 1$ e o tamanho do quorum é $|Q| = 2f + 1$. O sistema usa um esquema de *assinatura de limiar-(k,N)* com $k = f + 1$. Todos os clientes e servidores têm a chave pública do serviço. A chave privada está repartida por todos os servidores e é necessário um quorum de k para assinar um resultado do serviço, ou seja, um certificado.

O funcionamento básico do sistema é ilustrado pela figura 3.10. Um cliente envia um pedido para um dos servidores (o delegado) que o envia para outros $2f$, perfazendo $2f + 1$, ou seja, um quorum. Cada servidor obtém de alguma forma o certificado e assina-o com a sua parte da chave privada. Se ao fim de algum tempo o servidor não responder, o pedido é reenviado para outros $f + 1$ servidores de forma a que pelo menos um servidor correto o receba. Como cada quorum tem $2f + 1$ servidores, pelo menos $f + 1 = k$ deles irão devolver o certificado corretamente assinado, o que é suficiente para o cliente verificar a assinatura usando o esquema de assinatura de limiar-(k,N).

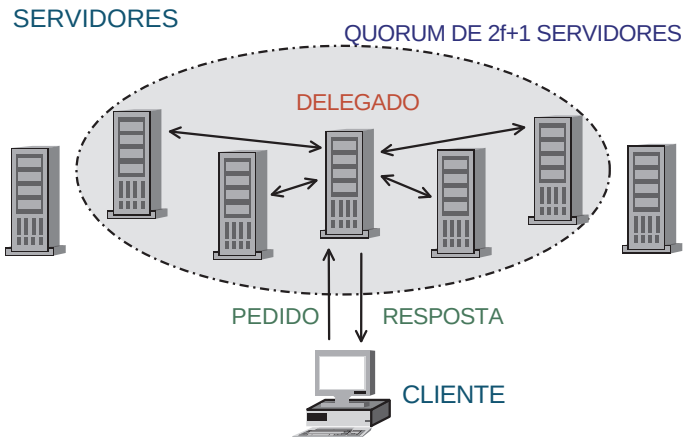

Figura 3.10. Funcionamento básico do COCA [Zhou et al., 2002a].

O esquema de recuperação proativa do COCA consiste em realizar periodicamente três operações:

1. renovar as partes das chaves privadas de cada servidor;
2. repor o código do servidor caso tenha sido corrompido;
3. repor o estado do servidor caso tenha sido corrompido.

Destas operações, a segunda e a terceira são semelhantes às do BFT-PR, sendo mais interessante explicar a primeira. A informação crítica que um atacante pode capturar se penetrar num servidor é a sua parte da chave privada, já que o COCA não assume que esta está em local seguro (ao contrário do BFT). Se o *adversário móvel* conseguisse capturar $f+1$ partes conseguiria personificar o serviço, logo a recuperação proativa tem de *renovar* essas partes.

Essa renovação das partes é baseada num *protocolo de partilha de segredos proativa* denominado APSS [Zhou et al., 2002b], que aliás é o primeiro protocolo assíncrono desse tipo. O protocolo é executado periodicamente, gerando de cada vez novas partes da chave privada. A chave privada, que se mantém sempre a mesma, nunca é materializada em nenhum dos servidores, tomando estes conhecimento apenas da sua parte. Um protocolo assíncrono de partilha de segredos proativa mais eficiente do que o APSS foi proposto em [Cachin et al., 2002].

Por fim, vale a pena referir que na sequência do COCA surgiu o sistema CODEX, uma evolução do COCA para armazenamento de dados [Marsh and Schneider, 2004]. A recuperação proativa usada no CODEX é semelhante à do COCA. Os dois sistemas têm um problema de modelo que fica claro através de duas afirmações contraditórias em [Zhou et al., 2002a]. Em diversos lugares do artigo é afirmado que o sistema é assíncrono mas, na seção 2.2, é também dito que “na prática, esperamos que possam ser feitas hipóteses temporais sobre partes do sistema que não tenham sido comprometidas” (pg. 334). Esta ambiguidade causa uma vulnerabilidade subtil que analisaremos de seguida.

3.5.3. Recuperação proativa em sistemas assíncronos?

Quando na seção 3.2.3 falámos de *modelos temporais*, referimos que o modelo assíncrono tinha a vantagem de não levantar hipóteses temporais, logo assumi-lo evita criar vulnerabilidades desse tipo. No entanto, recentemente foi mostrado que não é possível fazer recuperação proativa de forma segura (*safe*) em sistemas assíncronos [Sousa et al., 2005a, Sousa et al., 2005b].

A demonstração é elegante e vale a pena resumi-la.

Um protocolo ou algoritmo distribuído é sempre definido com base num conjunto de premissas. Exemplos de premissas incluem as que são feitos sobre o tipo e quantidade de faltas que podem ocorrer (p. ex. bizantinas e não mais do que f) e a sincronia da execução (p. ex. modelo assíncrono). Estas hipóteses são de fato uma abstração dos recursos de que o protocolo necessita para a sua execução. Por exemplo, quando se assume que não mais do que f servidores podem falhar, isso significa que o protocolo precisa de $N - f$ servidores corretos para funcionar corretamente.

O tipo de hipóteses que nos interessam são precisamente as deste tipo, as *hipóteses quantitativas de segurança sobre recursos*. Interessam-nos as hipóteses de segurança (*safety*), não as de progresso, pois é delas que depende a correção de um protocolo.

Dado um recurso qualquer *r*, esse recurso está *exausto* se tiver sido violada uma hipótese quantitativa de segurança sobre *r*. Um sistema diz-se *seguro-de-exaustão-r* (*r-exhaustion-safe*) se for capaz de garantir que o recurso *r* não será exausto.

Dado um sistema *A*, definimos $A_{t_{start}}$ como o instante de início da sua execução, $A_{t_{end}}$ o instante de terminação e $A_{t_{exaust}}$ o instante no qual determinado recurso *r* é exausto. Todos estes instantes são instantes de tempo real. O sistema *A* é seguro-de-exaustão-*r* sse $A_{t_{end}} < A_{t_{exaust}}$.

Consideremos um sistema composto por um serviço com vários servidores e clientes, como os que vimos acima. Esse sistema é *seguro-de-exaustão-servidor* se o número de servidores que podem falhar – no máximo f – não for exausto. Podemos dizer que existe um valor para o instante $A_{t_{exaust}}$ desse sistema, apesar de ser desconhecido e depender do "poder" do atacante. Como num sistema assíncrono os relógios podem derivar (*drift*) de forma ilimitada, e os tempos de processamento e comunicação são também ilimitados, não é possível garantir que o sistema terminará em determinado intervalo de tempo, ou seja, $A_{t_{end}}$ poderá tomar valores arbitrariamente grandes. Sendo assim, não é possível garantir que $A_{t_{end}} < A_{t_{exaust}}$ e o sistema não é seguro-de-exaustão-servidor.

O objetivo da recuperação proativa é garantir que nunca se atinge $A_{t_{exaust}}$ através da remoção das intrusões que possam ter ocorrido. Isso exige que a recuperação seja realizada *periodicamente* e que essa recuperação não demore mais do que um *tempo* determinado. Obviamente que para que isso seja possível é preciso levantar hipóteses *temporais* sobre o funcionamento do sistema, hipóteses essas que podem sempre ser violadas num sistema assíncrono. Na prática estas hipóteses podem ainda ser mais facilmente violadas se existirem ataques. Um atacante pode violar essas hipóteses, por exemplo, atrasando os relógios do sistema, ou atrasando a comunicação na rede (um ataque deste estilo ao sistema CODEX é explicado em [Sousa et al., 2005a]). Assim temos a impossibilidade

de recuperação proativa em sistemas estritamente assíncronos.

Uma solução para este problema consiste em usar um modelo de falhas híbrido baseado num *wormhole*, uma abordagem que já introduzimos atrás (v. seção 3.3.1.4). Ao contrário do resto do sistema, este componente deve ser síncrono de forma a que a recuperação possa ser executada a tempo. Como este componente é construído de forma a que seja seguro, as hipóteses de sincronia feitas em relação a ele não podem ser violadas devido a ataques. Convém notar que se continua a assumir que as intrusões nos servidores não ocorrem instantaneamente (no tempo real), mas de forma progressiva. O que esta solução permite garantir é que se não for possível num período $T_{rejuvenation}$ (tempo real) corromper $f+1$ servidores, então nunca serão corrompidos $f+1$ servidores e o sistema será *seguro-de-exaustão-servidor*.

3.6. Arquiteturas e sistemas

Apresentamos acima quatro tipos de soluções para tolerar intrusões: replicação de máquinas de estados, sistemas de quoruns, fragmentação e recuperação proativa. Como vimos, estes quatro tipos de mecanismos podem ser combinados de várias formas para concretizar *serviços distribuídos tolerantes a intrusões*, de que aliás vimos alguns exemplos. No entanto, existem outras técnicas que podemos considerar como sendo de *tolerância a intrusões* e que podem ser usadas para construir serviços distribuídos TI. Exemplos são a detecção de intrusões, a remoção dessas intrusões (quando ocorrem, não de forma proativa) e a reconfiguração. Outras técnicas de *prevenção* podem também ser usadas, como as *firewalls*. Nesta seção vamos ver diversas arquiteturas e sistemas da bibliografia que combinam diversos desses mecanismos para concretizar serviços TI. Voltando à nossa estória do pirata: o sistema pode ser bem mais complicado do que ter apenas um conjunto de torreões! Podem existir vigias para detectar intrusões, artilharia para disparar contra os piratas, etc.

A arquitetura comum a quase todos os sistemas que vimos até agora é a apresentada na figura 3.3. Duas exceções foram as arquiteturas da *secure store* (fig. 3.9) e a de separação de execução e acordo (fig. 3.6). Vimos ainda o modelo de falhas híbrido baseado numa arquitetura com um *wormhole* (seções 3.3.1.3 e 3.3.1.4).

3.6.1. Três variantes da arquitetura geral

Três arquiteturas ligeiramente mais complicadas do que a da figura 3.3 são comparadas em [Gupta et al., 2003].

A primeira arquitetura é chamada de *encaminhamento centralizado / gestão centralizada* e é apresentada na figura 3.11. O objetivo consiste em proteger um conjunto de servidores usando um pequeno número de componentes de confiança (*trusted*). Os pedidos recebidos são filtrados por uma *firewall*. Depois, chegam a uma *gateway* de confiança que os encaminha aleatoriamente para um dos servidores ativos. Cada servidor inclui um CMDaemon (*configuration management daemon*) encarregue de fazer detecção de intrusões localmente. Estes componentes informam um *gestor de configuração* centralizado e de confiança sobre o estado do servidor. O gestor limpa e recupera os servidores nos quais ocorram intrusões.

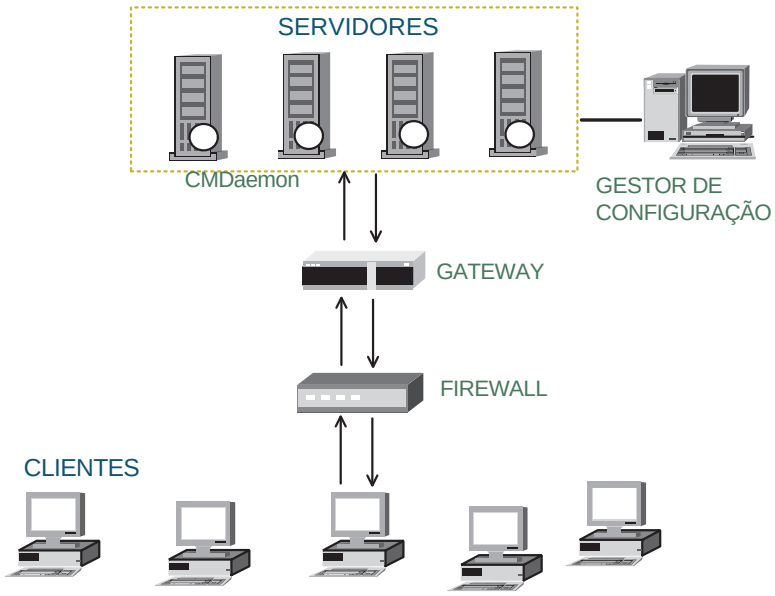

Figura 3.11. Arquitetura encaminhamento centralizado / gestão centralizada [Gupta et al., 2003].

A segunda arquitetura é denominada *encaminhamento por difusão / gestão centralizada* e tem muito em comum com a primeira. As principais diferenças são que não existe a *gateway* e que a *firewall* à entrada é substituída por uma *firewall* em cada servidor (figura 3.12). Nesta arquitetura cada servidor decide quais os pedidos que processa, p. ex. de acordo com uma política de balanceamento de carga.

Estas duas primeiras arquiteturas têm o inconveniente de poderem perder pedidos devido ao encaminhamento de mensagens para servidores com intrusões ainda não detectadas e removidas. No entanto, só são necessários $f + 1$ servidores e os pedidos são processados em paralelo o que, em princípio, permite obter melhor débito (pedidos processados por unidade de tempo).

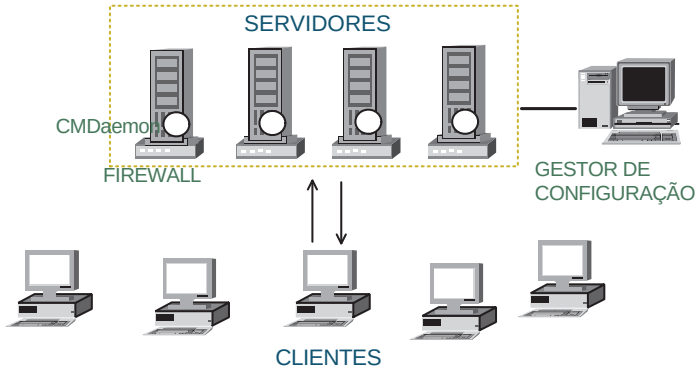

Figura 3.12. Arquitetura encaminhamento por difusão / gestão centralizada [Gupta et al., 2003].

A terceira arquitetura é chamada de *encaminhamento por difusão / gestão descentralizada* e difere da anterior pela não existência de um gestor de configuração (fig. 3.13). Esta arquitetura procura combinar o bom desempenho das duas anteriores com redundân-

cia de todos os componentes. Cada pedido é processado por apenas um dos servidores, mas os CMDaemons detectam intrusões e chegam a acordo sobre as reconfigurações que seja necessário realizar. A base desse mecanismo pode ser um sistema de comunicação em grupo, como o Rampart, sendo necessários $3f + 1$ servidores.

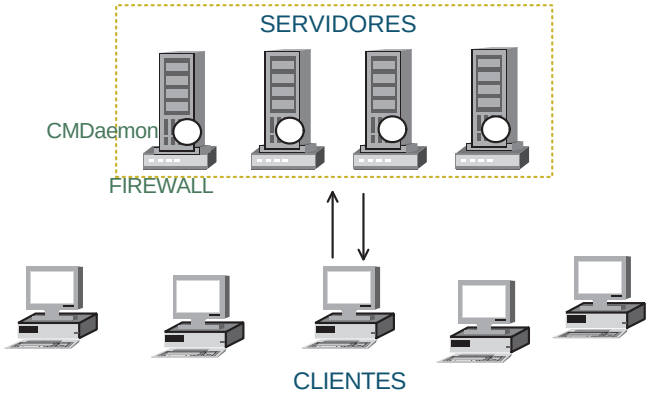

Figura 3.13. Arquitetura encaminhamento por difusão / gestão descentralizada [Gupta et al., 2003].

3.6.2. Arquitetura com *firewall* de privacidade

Voltemos à arquitetura que faz a separação entre servidores de execução e servidores de acordo. O artigo desta última arquitetura, apresenta uma versão mais complicada que não apresentamos na seção 3.6 por ser ortogonal ao tema aí discutido [Yin et al., 2003]. Esta outra arquitetura, que inclui uma *firewall de privacidade*, é apresentada na figura 3.14.

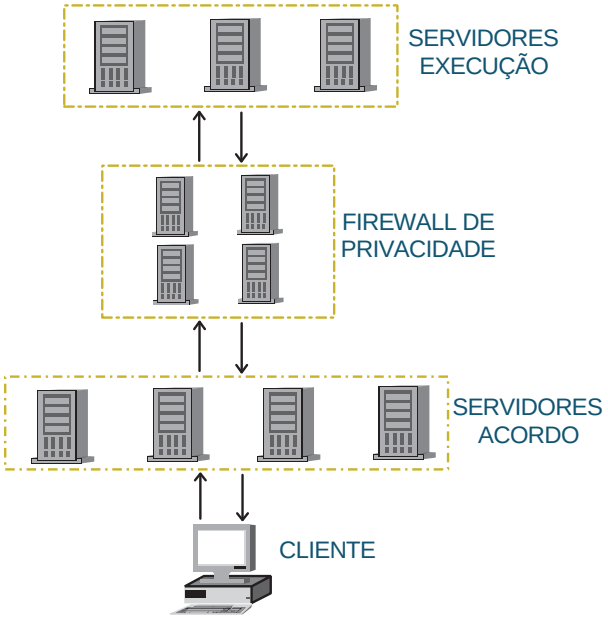

Figura 3.14. Arquitetura com separação execução-acordo e *firewall* de privacidade [Yin et al., 2003].

Como vimos atrás, um serviço baseado em replicação de máquinas de estados como o BFT não garante a confidencialidade da informação nos servidores pois um servi-

dor no qual tenha ocorrido uma intrusão pode difundir informação que aí se encontre. A *firewall* de privacidade pretende resolver este problema. Caso haja uma intrusão num *servidor de execução*, o componente filtra a informação que daí sai, garantindo assim a confidencialidade – ou privacidade caso se trate de informação pessoal. Só respostas bem formadas são deixadas passar por esse filtro. A idéia consiste em que a *firewall* tenha $h+1$ níveis, cada um com $h+1$ réplicas. Assim, é possível tolerar h intrusões nessas réplicas mantendo a disponibilidade, integridade e confidencialidade. Em outras palavras, são necessárias intrusões em $h+1$ dessas réplicas (e uma nos servidores de execução) para que o atacante possa obter informação confidencial.

3.6.3. Arquitetura SITAR

No âmbito do programa OASIS foi desenvolvida uma arquitetura abstrata denominada SITAR (*Scalable Intrusion-Tolerant Architecture for Distributed Systems*) com o objetivo de servir de base a serviços distribuídos TI [Wang et al., 2001]. A arquitetura pretende ser passível de ser introduzida entre clientes e servidores COTS (*commercial of-the-shelf*) sem que estes sejam modificados (v. fig. 3.15).

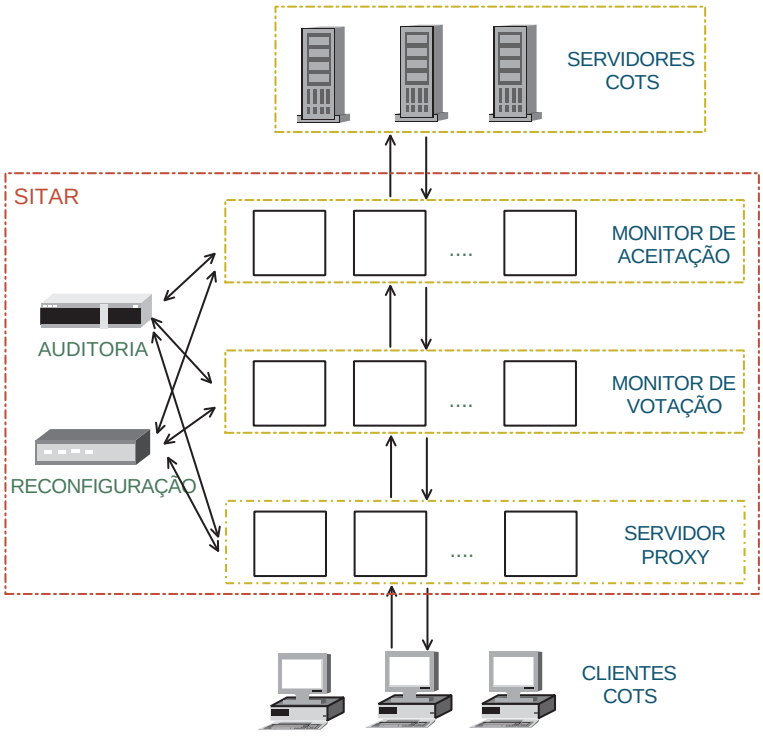

Figura 3.15. A arquitetura SITAR [Wang et al., 2001].

A arquitetura SITAR define um conjunto de módulos que podem ser instanciados de diferentes formas. O objetivo do servidor proxy é o de representar os servidores COTS perante os clientes COTS quando estes fazem pedidos. O monitor de aceitação e o monitor de votação, pelo contrário, atuam quando o servidores COTS respondem a um cliente. O monitor de aceitação faz algumas verificações de validade às respostas usando regras dependentes de cada serviço e faz detecção de intrusões nos servidores COTS. O monitor de votação recebe as respostas vindas do monitor de aceitação e faz votação para mascarar

intrusões. As respostas escolhidas são enviadas ao proxy que as reencaminha para os clientes.

3.6.4. Arquitetura DPASA

Quando o programa OASIS terminou, o DARPA criou um novo programa chamado OASIS Dem/Val para demonstrar e avaliar algumas das tecnologias aí pesquisadas. Nesse programa foi desenvolvida a arquitetura DPASA (*Designing Protection and Adaptation into a Survivability Architecture*) com o objetivo de demonstrar a TI num sistema real. A arquitetura DPASA é provavelmente a arquitetura de TI mais complexa e interessante desenvolvida até agora.

Os textos disponíveis sobre a arquitetura DPASA não descrevem propriamente a arquitetura mas antes a sua utilização numa aplicação que está a ser desenvolvida pela força aérea americana, a *Joint Battlespace Infosphere (JBI)* [Stevens et al., 2004, Atighetchi et al., 2005]. O objetivo de uma JBI é fornecer serviços de publicação-subscrição-interrogação (*publish-subscribe-query, PSQ*) a clientes de forma a que estes possam trocar informação sob a forma de objetos de informação (OIs).

O sistema IT-JBI – a versão TI da JBI – consiste num núcleo central que fornece serviços de comunicação a um conjunto de clientes. Uma versão simplificada da arquitetura do sistema encontra-se na figura 3.16.

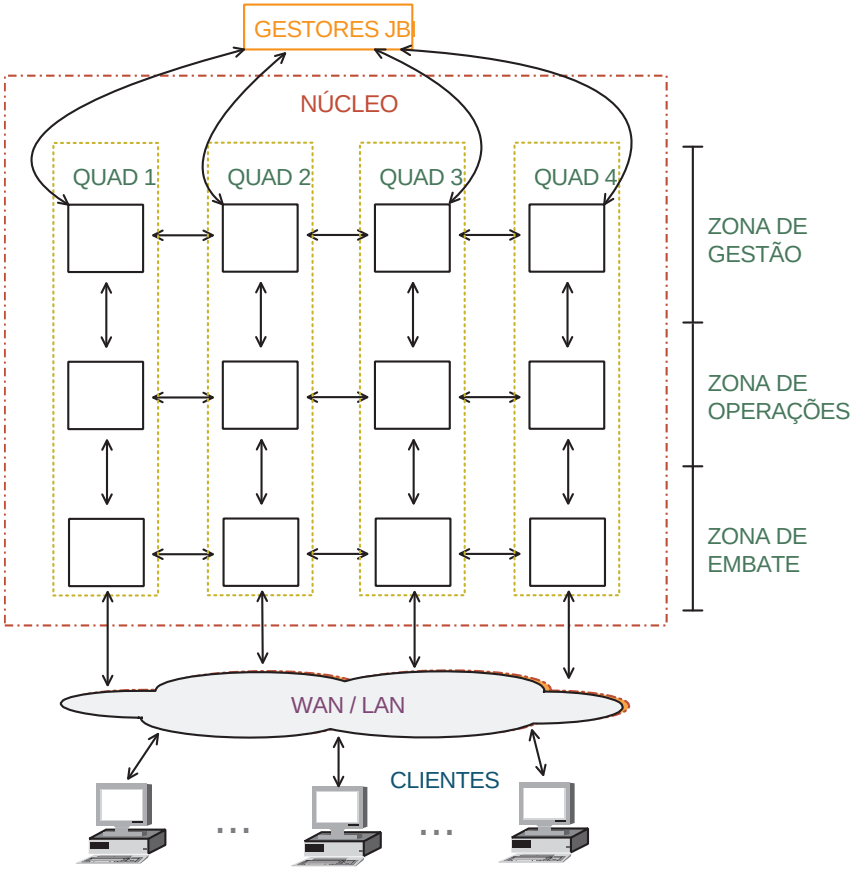

Figura 3.16. O sistema IT-JBI baseado na arquitetura DPASA [Stevens et al., 2004].

O núcleo é replicado em quatro quadrantes (*quad 1* a *quad 4*). As máquinas dos quatro quadrantes correm sistemas operacionais diferentes: SELinux, Solaris e Windows XP e 2000. Os adaptadores de rede das máquinas têm uma *firewall embutida* [Markham et al., 2003] que faz filtragem de pacotes e suporta redes privadas virtuais (VPNs) com máquinas com adaptadores do mesmo tipo.

O núcleo está também dividido em três zonas (v. figura). A *zona de embate* é a zona onde se dá o primeiro impato de um potencial ataque. A *zona de operações* é onde se encontra a concretização do serviço de publicação-subscrição-interrogação. A *zona de gestão* contém funções de gestão e controle do serviço. A comunicação entre zonas é controlada através de *switches* geridos e através de VPNs suportadas pelos adaptadores de rede.

A zona de embate contém um *proxy* de acesso (AP). O seu papel é fundamentalmente fazer de interface entre os protocolos e *middleware* de comunicação usados pelos clientes (p. ex. RMI, CORBA) e o JBI. A comunicação entre os clientes e os APs é feita através de VPNs de forma a evitar que o tráfego seja escutado na rede. Os APs têm dois adaptadores de rede. O adaptador exterior só recebe ligações vindas de clientes legítimos através de uma VPN e limita as portas de comunicação UDP e TCP. O adaptador interior serve para comunicar com os servidores da zona de operações.

A zona de operações contém os componentes que desempenham as principais funções do núcleo: os serviços de PSQ e a supervisão da segurança do núcleo e dos clientes a ele ligados. O processamento dos OIs é feito por três componentes desta zona: o servidor PSQ, o controlador do sentido descendente e o guardião. O papel do primeiro é evidente. O objetivo do segundo é verificar as assinaturas dos pedidos enviados pelos clientes para garantir a integridade desses mesmos pedidos. O guardião usa informação específica da aplicação para detectar OIs corrompidos.

A zona de gestão contém um componente que correlaciona a informação sobre intrusões obtida por sensores colocados nos clientes e nas zonas de tampão e operações. Essa informação é filtrada de forma a serem descartados alarmes falsos ou redundantes com base no contexto do sistema. Os alarmes importantes são enviados para outro componente da mesma zona denominada gestor de sistemas, SM. Este componente gera respostas às intrusões. O funcionamento do IT-JBI é supervisionado por gestores humanos (v. topo da figura).

O sistema IT-JBI foi testado no âmbito do programa OASIS Dem/Val por duas *red teams* com conhecimento total do sistema. O sistema resistiu bem aos ataques infligidos, exceto dois ataques de negação de serviço que o conseguiram bloquear temporariamente.

3.7. Problemas abertos

Depois do que tudo o que foi dito até agora, fica claro que a *tolerância a intrusões* já atingiu uma certa maturidade. Dos inúmeros exemplos que vimos é óbvio que já é possível construir *sistemas distribuídos tolerantes a intrusões* que ofereçam diversas propriedades e graus de segurança. Se aquilo que se pretende proteger for suficientemente valioso, já é possível criar um sistema de torreões dificilmente controlável pelo nosso pirata!

Apesar disso, a área da *tolerância a intrusões* está longe de estar fechada. Há

ainda um conjunto de problemas não triviais com os quais o arquiteto de sistemas tem de se defrontar. Esta seção irá listar os principais, dando pistas sobre a pesquisa que precisa de ser feita para resolver cada um deles.

3.7.1. Independência de faltas e diversidade

Todas as soluções de replicação e de fragmentação que vimos ao longo do capítulo partem sempre da mesma hipótese: é significativamente mais difícil penetrar em *m* servidores do que em um. Se o grau de dificuldade fosse idêntico, não valia a pena replicar ou fragmentar pois o atacante poderia tomar todos os servidores em vez de um só. Em outras palavras, tem de existir independência de faltas.

Quando se fala de intrusões o problema de garantir independência de faltas não é simples. Como vimos no modelo AVI, que caracteriza as faltas no domínio da *segurança* (fig. 3.2), uma intrusão é causada por um ou mais ataques que exploram com sucesso uma ou mais vulnerabilidades. Claramente a independência de faltas tem de ser garantida do lado das vulnerabilidades, já que é arriscado levantar hipóteses sobre os ataques. No entanto, se há algo que caracteriza as vulnerabilidades é que elas são problemas desconhecidos de um sistema computacional (salvo raras e desonrosas exceções). Logo, como vamos garantir que *N* servidores não sofrem do mesmo problema se nem sabemos do que é que sofre cada um deles?

Uma solução intuitiva é o recurso à *diversidade*. É evidente que os servidores devem ter sistemas operacionais diferentes, concretizações diferentes do código do sistema, etc. Este tipo de abordagens é usada em *tolerância a faltas*, p. ex. através do recurso à *programação com n versões* [Avizienis, 1985]. Também em *segurança* há algum debate sobre o tema, p. ex. em relação à monocultura MS-Windows na Internet e como esta favorece a difusão de código nocivo.

Diversos níveis de diversidade são sugeridos em [Deswarte et al., 1998, Castro and Liskov, 2002] e num estudo mais aprofundado sobre o tema apresentado em [Obelheiro et al., 2005]:

- diversidade de software da aplicação;
- diversidade de software de suporte à execução (sistema operacional, *middleware*, bases de dados, . . .);
- diversidade de hardware;
- diversidade de administradores dos servidores;
- diversidade de localização.

O significado dos diferentes níveis é bastante evidente. A diversidade do software da aplicação pode ser conseguida usando a já referida *programação com n versões*. A idéia consiste em ter *N* equipes de programadores que desenvolvem em paralelo versões de software com a mesma funcionalidade, uma para cada um dos servidores. Apesar do custo elevado que isso representa, em geral o custo de cada versão pode rondar apenas 0,7

a 0,85 o custo de uma versão de software normal, devido aos custos que não se multiplicam (estabelecimento de requisitos, modelagem do sistema) [Deswarte et al., 1998].

A conjugação de todos estes níveis de diversidade com técnicas de análise estática de código [Viega and McGraw, 2002] podem levar a níveis de confiança elevados mas qual é a independência de faltas que essa combinação realmente dá? Como garantir essa independência com determinado nível de confiança?

Outro tema interessante que merece ser pesquisado é a criação automática de diversidade. Será que é possível criar automaticamente diversas versões de software, possivelmente para diversos sistemas operacionais, com independência de vulnerabilidades?

Esta forma de diversidade entre as diversas réplicas apesar de tudo é simples quando comparada com outro tipo de diversidade desejável para a *recuperação proativa*: a diversidade num mesmo servidor após cada recuperação. A recuperação proativa parte do princípio que atacar cada servidor toma um determinado tempo mas isso não é bem verdade. O trabalho demorado é descobrir como se pode atacar um servidor; o processo de ataque em si geralmente é rápido se forem usado ferramentas automáticas (p. ex. um ataque *buffer overflow* demora . . .). Se um atacante ao fim de alguns dias descobrir como atacar $f + 1$ servidores e o ataque demorar alguns milisegundos, a recuperação proativa é inútil (o período típico de recuperação ronda um minuto). Uma solução seria que cada recuperação modificasse o servidor de tal forma que o atacante tivesse que recomeçar o processo de compreender como atacá-lo. Como conseguir isso?

3.7.2. Determinismo

A diversidade das réplicas no caso da *replicação de máquinas de estados* conduz a um problema complexo: o seu determinismo. Na seção 3.3.1 referimos a necessidade de que os servidores fossem determinísticos, ou seja, que o mesmo comando executado no mesmo estado inicial gerasse o mesmo estado final independentemente do servidor onde fosse executado. O problema é mais simples no caso dos sistemas de quoruns ou da fragmentação já que os servidores não executam comandos genéricos mas sobretudo armazenam dados em variáveis.

Uma experiência de concretização de um serviço distribuído TI relativamente simples mostrou como o problema do determinismo é particularmente relevante [Ferraz et al., 2004]. A idéia era concretizar um servidor de web TI usando replicação de máquinas de estados. As páginas consideradas estavam em HTML (não foi usado PHP ou ASPs) e não foi considerado o uso de HTTP-S, apenas HTTP. As réplicas eram idênticas (Linux e Apache). Mesmo com todas estas restrições o cabeçalho do HTTP tem diversos campos que não são determinísticos. Três campos geraram problemas: uma estampilha temporal com a data e a hora em que a página é retornada; um campo *Etag* que identifica univocamente a resposta ao pedido HTML; e um identificador do servidor. Muitos outros parâmetros podem gerar falta de determinismo, desde a representação de números reais às respostas a erros, passando por todo o tipo de operações que dependam do tempo.

Um sistema que pretende resolver este problema é uma evolução do BFT chamada BASE (*BFT with Abstract Specification Encapsulation*) [Castro et al., 2003]. O BASE

permite o uso de réplicas diferentes ou que não sejam deterministas. A solução consiste em fazer uma especificação abstrata do serviço, com um estado abstrato e um conjunto de operações que manipulam esse estado. Depois, concretiza-se um *wrapper* de conformidade que esconde cada réplica atrás de uma interface conforme com a especificação abstrata do serviço. Finalmente, caso seja necessário fazer transferência de estado entre réplicas, como no BFT, define-se uma função de abstração que traduz o estado de uma concretização para o estado abstrato e vice-versa.

O BASE resolve parte do problema mas deixa em aberto diversas questões. Como é que se lida com dados com estrutura complexa? E com dados comprimidos? E a principal questão: dado que em sistemas seguros muitas vezes é preciso cifrar a comunicação, como é que se lida com dados cifrados?

3.7.3. Detecção e remoção de intrusões

A detecção de intrusões pode ser considerada como fazendo parte da TI, já que é uma forma de *detecção de faltas* (seção 3.2.1). No entanto, a pesquisa em *sistemas de detecção de intrusões* é uma área com mérito reconhecido e que seria injusto reduzir a uma parte da TI.

Diversos sistemas TI têm englobado detecção de intrusões em soluções baseadas em replicação ou fragmentação, por exemplo, [Knight et al., 2001, Keromytis et al., 2003, Atighetchi et al., 2005]. No entanto os sistemas de detecção de intrusões continuam a ter uma elevada taxa de falsos positivos e falsos negativos o que na prática impede a resposta automática a intrusões. Mas se reduzirmos as intrusões que se pretendem detectar às que podem ocorrer num determinado tipo específico de serviço TI, não será possível fazer essa detecção com precisão suficiente para fazer resposta automática?

3.7.4. Avaliação

Em *sistemas distribuídos* existem diversas métricas que permitem comparar dois algoritmos que resolvem o mesmo problema. As métricas mais usadas são o número (ou a complexidade) de ciclos de comunicação e o número (ou a complexidade) de mensagens trocadas. As comparações podem ser matizadas (*o algoritmo X é melhor do que o Y no caso Z mas não no caso W*) mas em geral essas métricas permitem avaliar algoritmos sem ambiguidade.

Em *confiança no funcionamento* foram também desenvolvidos diversos métodos para avaliar a tolerância a faltas de sistemas e algoritmos. Um exemplo são os métodos estocásticos que permitem determinar a probabilidade de um sistema falhar dada a probabilidade de ocorrerem faltas em certos componentes e dessas faltas se propagarem [Nicol et al., 2004].

Em *tolerância a intrusões* é igualmente importante avaliar a segurança de sistemas e algoritmos. Alguns trabalhos recentes têm feito esse tipo de análise usando métodos estocásticos [Gupta et al., 2003, Singh et al., 2003, Stevens et al., 2004] e redes de Petri [Wang et al., 2003]. No entanto, essas avaliações assumem comportamentos probabilísticos dos ataques e intrusões o que não parece ser realista. O ideal seria existirem métricas simples que permitissem avaliar esses sistemas como é feito com os algoritmos distribuídos.

3.7.5. Negação de serviço e privacidade

Os ataques de negação de serviço (DoS) e as questões de privacidade em sistemas TI têm sido pouco tratadas na bibliografia e são geralmente importantes em sistemas reais.

Os ataques DoS ganharam fama mundial com o grande ataque de Fevereiro de 2000 que deixou serviços como o Yahoo, CNN e eBay inoperantes durante horas. Estes ataques consistiram em usar aquilo que hoje se chamaria uma *botnet* para bombardear esses serviços com tráfego. O problema é tanto mais grave quanto maior for o processamento feito por cada pedido. No caso dos serviços TI é evidente que cada pedido exige algum processamento: os pedidos são executados por vários servidores, são feitas diversas operações criptográficas, ...

Como lidar com ataques DoS em serviços TI? Uma solução baseada numa rede *overlay* é usada no sistema SABER [Keromytis et al., 2003]. A idéia consiste em usar a porção da rede que cerca o servidor para filtrar agressivamente o tráfego que lhe é dirigido com base no endereço de origem. Em cada momento existe um número limitado de endereços de origem aceites, de que só os clientes válidos devem ter conhecimento. A idéia é interessante mas parece necessitar de um número elevado de nós na rede *overlay*. Também o sistema COCA procura lidar com este tipo de ataque usando um conjunto de mecanismos bastante simples [Zhou et al., 2002a]. O que nenhum desses trabalhos considera é a relação entre serviços que têm por objetivo garantir, entre outras propriedades, a disponibilidade, e ataques realizados precisamente contra essa disponibilidade, que é o que constituem os ataques DoS. Será que não seria possível aproveitar as próprias características dos serviços distribuídos TI para combater esses ataques? Qual a arquitetura adequada a esse fim?

A questão da privacidade não tem sido tratada nos trabalhos sobre serviços TI com a exceção de [Yin et al., 2003]. É evidente que as soluções que permitem obter confidencialidade (seção 3.4) também contribuem para a privacidade de dados pessoais aí armazenados. No entanto, a privacidade face ao próprio administrador do serviço não é garantida. Soluções?

3.8. Conclusão

Uma vez terminada a exposição sobre *serviços distribuídos tolerantes a intrusões* – conceitos básicos, mecanismos, arquiteturas, problemas abertos – é interessante opinar sobre a utilização que será dada a estes resultados de vários anos de pesquisa. A questão não é certamente a necessidade de serviços seguros, que é evidente. O que nos podemos perguntar é se (1) a TI permite obter essa segurança, (2) a TI está pronta para ser usada já hoje, e (3) em que tipo de aplicações será realmente usada.

A resposta à primeira questão é positiva. Obviamente, como tão bem ilustra o sistema IT-JBI, uma arquitetura TI completa inclui numerosos componentes e mecanismos que vêm da *segurança* clássica, não são nenhuma novidade da TI. No entanto a possibilidade de multiplicar por N (ou $f+1$ ou ...) a dificuldade de atacar e controlar um serviço traz claramente segurança acrescida.

Quanto à segunda questão, a resposta é também positiva. Existem lacunas que podem ser pesquisadas e a TI tornada muito mais prática, mas as soluções que vimos são

reais e utilizáveis já hoje.

Algumas pistas sobre a terceira questão podem vir da forma como a *tolerância a faltas* é usada hoje em dia. Muitas soluções complexas e caras são usadas em aplicações que justificam o seu custo, p. ex. por envolverem risco de vidas humanas. Exemplos incluem a aviação, a indústria espacial e centrais de produção de energia. No entanto, algumas soluções tolerantes a faltas mais simples estão disponíveis um pouco por toda a parte: desde cabos de telecomunicações replicados a soluções tolerantes a faltas em discos rígidos (RAID), passando pela detecção de erros na memória dos PCs comuns.

Em relação à *tolerância a intrusões* podemos esperar que algo de semelhante aconteça. Algumas soluções complexas e caras serão certamente usadas em aplicações críticas, nem que o sejam só sob o ponto de vista financeiro. Exemplos que ocorrem são a rede SWIFT, que interliga mais de 7000 bancos de todo o mundo e que movimenta triliões de euros todos os dias, ou aplicações militares como o IT-JBI. Na realidade, já há alguns anos que a chave de raiz do sistema SET da MasterCard/VISA se encontrava distribuída por diversas empresas diferentes usando criptografia de limiar [Frankel and Yung, 1998]. Além disso, muitas soluções simples poderão ser usadas para tornar mais seguros componentes de sistemas de menor custo. Por exemplo, com o sucesso das soluções de armazenamento de dados em rede (NAS, SAN) poderão surgir produtos comerciais que forneçam disponibilidade, integridade e confidencialidade sem grande desperdício de espaço de disco graças ao uso de códigos de apagamento.

A *tolerância a intrusões* não é certamente a panaceia para todos os problemas de segurança dos sistemas computacionais do nosso tempo. No entanto fornece um conjunto de soluções válidas que podem certamente contribuir para a segurança de algumas fortalezas nesse mundo dominado por piratas em que se tornou a Internet.

Agradecimentos

Este capítulo surge após seis anos de pesquisa em *tolerância a intrusões*. Durante esse tempo trabalhei sempre com duas pessoas a quem estou agradecido por muitas razões: Paulo Veríssimo e Nuno Ferreira Neves. Trabalhei também com vários outros colegas dos quais não posso deixar de mencionar um: Lau Cheuk Lung. O Alysson Bessani Neves e o Paulo Sousa tiveram a paciência de fazer uma revisão do manuscrito e dar muitas sugestões de melhoramentos, fato pelo qual estou muito agradecido.

Referências

[Adelsbach et al., 2002] Adelsbach, A., Alessandri, D., Cachin, C., Creese, S., Deswarte, Y., Kursawe, K., Laprie, J. C., Powell, D., Randell, B., Riordan, J., Ryan, P., Simmonds, W., Stroud, R., Veríssimo, P., Waidner, M., and Wespi, A. (2002). *Conceptual Model and Architecture of MAFTIA. Project MAFTIA deliverable D21*.

[Alon et al., 2000] Alon, N., Kaplan, H., Krivelevich, M., Malkhi, D., and Stern, J. (2000). Scalable secure storage when half the system is faulty. In Montanari, U., Rolim, J., and Welzl, R., editors, *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 576–587. Springer-Verlag.

[Atighetchi et al., 2005] Atighetchi, M., Rubel, P., Pal, P., Chong, J., and Studin, L. (2005). Case study: The intrusion tolerant JBI. Technical report, BBN Technologies.

[Avizienis, 1985] Avizienis, A. (1985). The N-version approach to fault tolerant software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501.

[Avizienis et al., 2004] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

[Baldoni et al., 2000] Baldoni, R., Helary, J., Raynal, M., and Tanguy, L. (2000). Consensus in Byzantine asynchronous systems. In *Proceedings of the International Colloquium on Structural Information and Communication Complexity*, pages 1–16.

[Bazzi and Ding, 2004] Bazzi, R. and Ding, Y. (2004). Non-skipping timestamps for Byzantine data storage systems. In Guerraoui, R., editor, *Proceedings of the 18th International Conference on Distributed Computing*, volume 3274 of *Lecture Notes in Computer Science*, pages 405–419. Springer-Verlag.

[Ben-Or, 1983] Ben-Or, M. (1983). Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30.

[Bessani et al., 2005] Bessani, A. N., da Silva Fraga, J., and Lung, L. C. (2005). O confeiteiro bizantino: Exclusão mútua em sistemas abertos sujeitos a faltas bizantinas. In *Anais do 23o. Simpósio Brasileiro de Redes de Computadores*.

[Blakley, 1979] Blakley, G. R. (1979). Safeguarding cryptographic keys. In *Proceedings of the AFIPS National Computer Conference*, volume 48, pages 313–317.

[Bracha, 1984] Bracha, G. (1984). An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 154–162.

[Bracha and Toueg, 1985] Bracha, G. and Toueg, S. (1985). Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840.

[Cachin, 2002] Cachin, C. (2002). Personal communication.

[Cachin et al., 2002] Cachin, C., Kursawe, K., Lysyanskaya, A., and Strobl, R. (2002). Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97.

[Cachin et al., 2001] Cachin, C., Kursawe, K., Petzold, F., and Shoup, V. (2001). Secure and efficient asynchronous broadcast protocols (extended abstract). In Kilian, J., editor, *Advances in Cryptology: CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer-Verlag.

[Cachin et al., 2000] Cachin, C., Kursawe, K., and Shoup, V. (2000). Random oracles in Contanstinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 123–132.

[Cachin and Poritz, 2002] Cachin, C. and Poritz, J. A. (2002). Secure intrusion-tolerant replication on the Internet. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 167–176.

[Cachin and Tessaro, 2004] Cachin, C. and Tessaro, S. (2004). Asynchronous verifiable information dispersal. RZ 3569, IBM Research.

[Cachin and Tessaro, 2005] Cachin, C. and Tessaro, S. (2005). Optimal resilience for erasure-coded Byzantine distributed storage. RZ 3575, IBM Research.

[Canetti et al., 1997] Canetti, R., Gennaro, R., Herzberg, A., and Naor, D. (1997). Proactive security: Long-term protection against break-ins. *RSA CryptoBytes*, 3(1):1–8.

[Castro and Liskov, 2002] Castro, M. and Liskov, B. (2002). Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461.

[Castro et al., 2003] Castro, M., Rodrigues, R., and Liskov, B. (2003). BASE: Using abstraction to improve fault tolerance. *ACM Transactions Computer Systems*, 21(3):236–269.

[CERT/CC, 2005] CERT/CC (2005). CERT coordination center statistics 1988-2005. http://www.cert.org/stats/.

[Chandra and Toueg, 1996] Chandra, T. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

[Correia et al., 2002a] Correia, M., Lung, L. C., Neves, N. F., and Veríssimo, P. (2002a). Efficient Byzantine-resilient reliable multicast on a hybrid failure model. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, pages 2–11.

[Correia et al., 2005a] Correia, M., Neves, N. F., Lung, L. C., and Veríssimo, P. (2005a). Low complexity Byzantine-resilient consensus. *Distributed Computing*, 17(3):237–249.

[Correia et al., 2005b] Correia, M., Neves, N. F., Lung, L. C., and Veríssimo, P. (2005b). Worm-IT – a wormhole-based intrusion-tolerant group communication system. Submitted for publication.

[Correia et al., 2004] Correia, M., Neves, N. F., and Veríssimo, P. (2004). How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, pages 174–183.

[Correia et al., 2002b] Correia, M., Veríssimo, P., and Neves, N. F. (2002b). The design of a COTS real-time distributed security kernel. In *Proceedings of the Fourth European Dependable Computing Conference*, pages 234–252.

[Deswarte et al., 1991] Deswarte, Y., Blain, L., and Fabre, J. C. (1991). Intrusion tolerance in distributed computing systems. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 110–121.

[Deswarte et al., 1998] Deswarte, Y., Kanoun, K., and Laprie, J. C. (1998). Diversity against accidental and deliberate faults. In *Computer Security, Dependability, & Assurance: From Needs to Solutions*. IEEE Press.

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

[Doudou et al., 2002] Doudou, A., Garbinato, B., and Guerraoui, R. (2002). Encapsulating failure detection: From crash-stop to Byzantine failures. In *International Conference on Reliable Software Technologies*, pages 24–50.

[Doudou and Schiper, 1997] Doudou, A. and Schiper, A. (1997). Muteness detectors for consensus with Byzantine processes. Technical Report 97/30, EPFL.

[Dwork et al., 1988] Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323.

[Ferraz et al., 2004] Ferraz, R., Goncalves, B., Sequeira, J., Correia, M., Neves, N. F., and Veríssimo, P. (2004). An intrusion-tolerant web server based on the DISTRACT architecture. In *Proceedings of the Workshop on Dependable Distributed Data Management*.

[Fischer et al., 1985] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382.

[Fraga and Powell, 1985] Fraga, J. S. and Powell, D. (1985). A fault- and intrusion-tolerant file system. In *Proceedings of the 3rd International Conference on Computer Security*, pages 203–218.

[Frankel and Yung, 1998] Frankel, Y. and Yung, M. (1998). Risk management using theshold RSA cryptosystems. *Usenix ;login: online*.

[Garay et al., 2000] Garay, J. A., Gennaro, R., Jutla, C., and Rabin, T. (2000). Secure distributed storage and retrieval. *Theoretical Computer Science*, 243(1-2):363–389.

[Gemmell, 1997] Gemmell, P. S. (1997). An introduction to threshold cryptography. *Cryptobytes*, 2(3):7–12.

[Gifford, 1979] Gifford, D. K. (1979). Weighted voting for replicated data. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 150–162.

[Goodson et al., 2004] Goodson, G., Wylie, J., Ganger, G., and Reiter, M. (2004). Efficient Byzantine-tolerant erasure-coded storage. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*.

[Gupta et al., 2003] Gupta, V., Lam, V., Ramasamy, H., Sanders, W., and Singh, S. (2003). Dependability and performance evaluation of intrusion-tolerant server architectures. In *Proceedings of the First Latin-American Symposium on Dependable Computing*, pages 81–101.

[Hadzilacos and Toueg, 1994] Hadzilacos, V. and Toueg, S. (1994). A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, Department of Computer Science.

[Herlihy and Wing, 1990] Herlihy, M. P. and Wing, J. (1990). Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492.

[Keromytis et al., 2003] Keromytis, A., Gross, P., Kaiser, G., Misra, V., Nieh, J., Rubenstein, D., and Stolfo, S. (2003). A holistic approach to service survivability. In *Proceedings of the ACM Workshop on Survivable and Self-Regenerative Systems*, pages 11–22.

[Kihlstrom et al., 2001] Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (2001). The SecureRing group communication system. *ACM Transactions on Information and System Security*, 4(4):371–406.

[Kihlstrom et al., 2003] Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (2003). Byzantine fault detectors for solving consensus. *The Computer Journal*, 46(1):16–35.

[Knight et al., 2001] Knight, J., Heimbigner, D., Wolf, A., Carzaniga, A., Hill, J., and Devanbu, P. (2001). The Willow survivability architecture. In *Proceedings of the 4th Information Survivability Workshop*.

[Krawczyk, 1993] Krawczyk, H. (1993). Distributed fingerprints and secure information dispersal. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, pages 207–218.

[Lakshmanan et al., 2003] Lakshmanan, S., Ahamad, M., and Venkateswaran, H. (2003). Responsive security for stored data. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):818–828.

[Lala, 2003] Lala, J. H., editor (2003). *Foundations of Intrusion Tolerant Systems*. IEEE Computer Society Press.

[Lamport, 1978] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565.

[Lamport, 1986] Lamport, L. (1986). On interprocess communication (part II: Algorithms). *Distributed Computing*, 1:86–101.

[Lamport et al., 1982] Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401.

[Lippmann et al., 2000] Lippmann, R., Haines, J., Fried, D., Korba, J., and Das, K. (2000). Analysis and results of the 1999 DARPA off-line intrusion detection evaluation. In Debar, H., Mé, L., and Wu, S. F., editors, *Recent Advances in Intrusion Detection - Third International Workshop*, volume 1907 of *Lecture Notes in Computer Science*, pages 162–182. Springer-Verlag.

[Malkhi and Reiter, 1997a] Malkhi, D. and Reiter, M. (1997a). Byzantine quorum systems. In *Proceedings of the 29th ACM Symposium in Theory of Computing*, pages 569–578.

[Malkhi and Reiter, 1997b] Malkhi, D. and Reiter, M. (1997b). Unreliable intrusion detection in distributed computations. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 116–124.

[Malkhi and Reiter, 1998a] Malkhi, D. and Reiter, M. (1998a). Byzantine quorum systems. *Distributed Computing*, 11:203–213.

[Malkhi and Reiter, 1998b] Malkhi, D. and Reiter, M. (1998b). Secure and scalable replication in Phalanx. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*.

[Malkhi et al., 1997] Malkhi, D., Reiter, M., and Wool, A. (1997). The load and availability of Byzantine quorum systems. In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing*, pages 249–257.

[Malkhi and Reiter, 2000] Malkhi, D. and Reiter, M. K. (2000). An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):187–202.

[Markham et al., 2003] Markham, T., Meredith, L., and Payne, C. (2003). Distributed embedded firewalls with virtual private groups. In *DARPA Information Survivability Conference and Exposition - Volume II*.

[Marsh and Schneider, 2004] Marsh, M. A. and Schneider, F. B. (2004). CODEX: A robust and secure secret distribution system. *IEEE Transactions on Dependable and Secure Computing*, 1(1):34–47.

[Martin and Alvisi, 2004] Martin, J. P. and Alvisi, L. (2004). A framework for dynamic Byzantine storage. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, pages 325–334.

[Martin and Alvisi, 2005] Martin, J. P. and Alvisi, L. (2005). Fast Byzantine consensus. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*.

[Martin et al., 2002a] Martin, J. P., Alvisi, L., and Dahlin, M. (2002a). Minimal Byzantine storage. In *Proceedings of the 16th International Conference on Distributed Computing*, volume 2508 of *LNCS*, pages 311–325. Springer-Verlag.

[Martin et al., 2002b] Martin, J. P., Alvisi, L., and Dahlin, M. (2002b). Small Byzantine quorum systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 374–383.

[Menezes et al., 1997] Menezes, A. J., Oorschot, P. C. V., and Vanstone, S. A. (1997). *Handbook of Applied Cryptography*. CRC Press.

[Moser and Melliar-Smith, 1999] Moser, L. E. and Melliar-Smith, P. M. (1999). Byzantine-resistant total ordering algorithms. *Information and Computation*, 150:75–111.

[Moser et al., 2000] Moser, L. E., Melliar-Smith, P. M., and Narasimhan, N. (2000). The SecureGroup communication system. In *Proceedings of the IEEE Information Survivability Conference*, pages 507–516.

[Naor and Wool, 1996] Naor, M. and Wool, A. (1996). Access control and signatures via quorum secret sharing. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 157–168.

[Neves et al., 2005] Neves, N. F., Correia, M., and Veríssimo, P. (2005). Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems*. Accepted for publication.

[Nicol et al., 2004] Nicol, D. M., Sanders, W. H., and Trivedi, K. S. (2004). Model-based evaluation: From dependability to security. *IEEE Transactions on Dependable and Secure Computing*, 1(1):48–65.

[Obelheiro et al., 2005] Obelheiro, R. R., Bessani, A. N., Fraga, J. S., and Lung, L. C. (2005). Analisando a viabilidade da implementação prática de sistemas tolerantes a intrusões. In *Anais do 5o Simpósio Brasileiro de Segurança*.

[Ostrovsky and Yung, 1991] Ostrovsky, R. and Yung, M. (1991). How to withstand mobile virus attacks. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 51–59.

[Rabin, 1983] Rabin, M. O. (1983). Randomized Byzantine generals. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 403–409.

[Rabin, 1989] Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348.

[Ramasamy et al., 2002] Ramasamy, H., Pandey, P., Lyons, J., Cukier, M., and Sanders, W. H. (2002). Quantifying the cost of providing intrusion tolerance in group communication systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 229–238.

[Reiter, 1994] Reiter, M. (1994). Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80.

[Reiter, 1995] Reiter, M. K. (1995). The Rampart toolkit for building high-integrity services. In *Theory and Practice in Distributed Systems*, volume 938 of *Lecture Notes in Computer Science*, pages 99–110. Springer-Verlag.

[Rivest et al., 1978] Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[Schneider, 1990] Schneider, F. B. (1990). Implementing faul-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

[Shamir, 1979] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(1):612–613.

[Singh et al., 2003] Singh, S., Cukier, M., and Sanders, W. H. (2003). Probabilistic validation of an intrusion-tolerant replication system. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 615–624.

[Sousa et al., 2005a] Sousa, P., Neves, N. F., and Verissimo, P. (2005a). How resilient are distributed f fault/intrusion-tolerant systems? In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*.

[Sousa et al., 2005b] Sousa, P., Neves, N. F., and Veríssimo, P. (2005b). Proactive resilience through architectural hybridization. DI/FCUL TR 05–8, Department of Informatics, University of Lisbon.

[Stevens et al., 2004] Stevens, F., Courtney, T., Singh, S., Agbaria, A., Meyer, J. F., Sanders, W. H., and Pal, P. (2004). Model-based validation of an intrusion-tolerant information system. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, pages 184–194.

[Turner et al., 2004] Turner, D., Entwisle, S., Friedrichs, O., Hanson, D., Fossi, M., Ahmad, D., Gordon, S., Szor, P., and Chien, E. (2004). Symantec Internet security threat report. Trends for January 1, 2004 – June 30, 2004. Volume VI.

[Veríssimo, 2003] Veríssimo, P. (2003). Uncertainty and predictability: Can they be reconciled? In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 108–113. Springer-Verlag.

[Veríssimo and Casimiro, 2002] Veríssimo, P. and Casimiro, A. (2002). The Timely Computing Base model and architecture. *IEEE Transactions on Computers*, 51(8):916–930.

[Veríssimo and de Lemos, 1989] Veríssimo, P. and de Lemos, R. (1989). Confiança no funcionamento: Proposta para uma terminologia em português. Technical Report RT48-89, INESC.

[Veríssimo et al., 2000] Veríssimo, P., Neves, N. F., and Correia, M. (2000). The middleware architecture of MAFTIA: A blueprint. In *Proceedings of the Third IEEE Information Survivability Workshop*.

[Veríssimo et al., 2003] Veríssimo, P., Neves, N. F., and Correia, M. (2003). Intrusion-tolerant architectures: Concepts and design. In Lemos, R., Gacek, C., and Romanovsky, A., editors, *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*, pages 3–36. Springer-Verlag.

[Veríssimo and Rodrigues, 2001] Veríssimo, P. and Rodrigues, L. (2001). *Distributed Systems for System Architects*. Kluwer Academic Publishers.

[Viega and McGraw, 2002] Viega, J. and McGraw, G. (2002). *Building Secure Software*. Addison Wesley.

[Wang et al., 2003] Wang, D., Madan, B., and Trivedi, K. (2003). Security analysis of the SITAR intrusion tolerance system. In *Proceedings of the ACM Workshop on Survivable and Self-Regenerative Systems*, pages 23–32.

[Wang et al., 2001] Wang, F., Gong, F., Sargor, C., Goseva-Popstojana, K., Trivedi, K., and Jou, F. (2001). SITAR: A scalable intrusion tolerance architecture for distributed services. In *Proceedings of the IEEE Second SMC Information Assurance Workshop*, pages 38–45.

[Yin et al., 2003] Yin, J., Martin, J., Venkataramani, A., Alvisi, L., and Dahlin, M. (2003). Separating agreement from execution for Byzantine fault tolerant services. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 253–267.

[Zhou et al., 2002a] Zhou, L., Schneider, F., and van Renesse, R. (2002a). COCA: A secure distributed on-line certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368.

[Zhou et al., 2002b] Zhou, L., Schneider, F., and van Renesse, R. (2002b). Proactive secret sharing in asynchronous systems. TR 1877, Cornell University.