COMPARAÇÃO ENTRE FILTROS DE PACOTES COM ESTADOS E TECNOLOGIAS TRADICIONAIS DE FIREWALL

Marcelo Barbosa Lima
Instituto de Computação
Universidade Estadual de Campinas
13083-970 Campinas – SP
marcelo.lima@dcc.unicamp.br

Paulo Lício de Geus
Instituto de Computação
Universidade Estadual de Campinas
13083-970 Campinas - SP
paulo@dcc.unicamp.br

RESUMO

Este trabalho compara as vantagens e desvantagens de filtros de pacotes com estados (Stateful Packet Filter) em relação às tecnologias tradicionais de firewall: filtros de pacotes e proxies. Será mostrado aqui, que um filtro de pacotes com estados é uma solução que em alguns casos (pequenas Intranets onde os riscos são pequenos e redes locais onde o desempenho é muito importante), poderia substituir de maneira segura as tecnologias tradicionais de firewall (sem a necessidade de filtros de pacotes e agentes proxy). Entretanto, em ambientes onde os requisitos de segurança são maiores, proxies específicos para determinados protocolos e serviços críticos devem ser adicionados à filtragem.

ABSTRACT

This work compares the advantages and disadvantages of stateful packet filters in relation to the traditional technologies of firewall: packet filters and proxies. It will be shown here, that a stateful packet filter is a solution that in some cases (little Intranets where the risks are small and local networks where the throughput is very important), could safely to substitute traditional technologies of firewall (without no need for packet filters and proxy agents). However, in rooms where security requirements are higher, specific proxies for certain critical protocols and services should be added to the filtering.

1 INTRODUÇÃO

Nos últimos anos, a indústria de segurança na Internet tem crescido rapidamente e vários produtos estão disponíveis na classe de *firewalls*, gerando uma certa confusão para quem procura uma solução que melhor se enquadre às suas necessidades de segurança.

Neste artigo, é feita uma comparação entre as diferentes tecnologias de *firewall* que fornecem a base usada na implementação desses produtos, ajudando a esclarecer as suas características importantes, vantagens e desvantagens, ajudando portanto a descobrir qual produto é o mais conveniente, dependendo dos requisitos de segurança desejados. Alguns produtos são híbridos, ou seja, apresentam características de tecnologias diferentes. Além das tradicionais, filtros de pacotes e agentes *proxy*, uma nova alternativa (se é que pode ser considerada nova, uma vez que para muitos, não passa de uma nova geração de filtros de pacotes – Avolio, 1998) está sendo usada: filtros de pacotes com estados (*stateful packet filter*, SPF), que já obtém boa parcela do mercado de *firewalls* e tem o *Firewall-1* da *CheckPoint* como seu principal representante. Será mostrado que filtros de pacotes com estados podem ser suficientes em alguns ambientes de computação, mas em outros, onde os requisitos de segurança são maiores para alguns serviços, será necessário o uso de *proxies* específicos para estes serviços, possivelmente complementando a atuação do SPF.

A seção 2 apresenta as tecnologias tradicionais (filtros de pacotes e *proxies*), com suas principais vantagens e deficiências. A seção 3 apresenta filtros de pacotes com estados, como eles tentam aproveitar algumas das principais vantagens de filtros de pacotes e de *proxies* e as suas desvantagens em relação a um *proxy* dedicado a um determinado protocolo ou serviço. A seção 4 mostra que um filtro com estados pode ser uma solução suficientemente segura em certos ambientes, mas *proxies* podem ser necessários (em adição à filtragem) em outros onde os requisitos de segurança sejam maiores. A seção 5 vem com uma breve discussão sobre limitações de *firewalls*. A seção 6 traz as considerações finais e as conclusões deste trabalho.

2 TECNOLOGIAS TRADICIONAIS

Nesta seção veremos as duas tecnologias tradicionais de *firewalls* (Cheswick e Bellovin, 1994; Chapman, e Zwick, 1995; Sponph, 1997; Gonçalves, 1997; Pouw, 1999), que são os filtros de pacotes e *proxies*.

2.1 Filtros de Pacotes

Filtragem de pacotes é o processo de permitir ou evitar tráfego de pacotes entre redes com base nas informações existentes nos cabeçalhos de cada pacote e em um conjunto de regras de filtragem. Em produtos que implementam filtragem de pacotes as informações utilizadas são aquelas existentes nos cabeçalhos dos níveis de rede e de transporte de cada pacote.

Por diversos anos, várias redes locais fizeram uso de roteadores filtradores ou escrutinadores (*screening routers*) para garantir a segurança de suas informações e controlar acesso. Estes roteadores implementavam o processo de filtragem juntamente com o processo de roteamento de

pacotes. Esta arquitetura de *firewall* com um único roteador filtrador pode ser bastante atraente, uma vez que estes hardware e software de roteamento já são mesmo necessários para que a rede local seja conectada no nível de rede.

Por concentrar-se nas informações das camadas de rede e de transporte, um filtro de pacotes pode ser implementado também no núcleo do sistema operacional, o que lhe garante sempre um bom desempenho. Além disso, por não depender do nível de aplicação, tem ainda como vantagens a transparência (não precisa de configurações especiais nos clientes) e a escalabilidade (novos serviços são facilmente suportados).

Estes primeiros filtros de pacotes, por usarem regras de filtragem estáticas, são classificados como filtros de pacotes estáticos. Tais filtros de pacotes não conseguem tratar convenientemente serviços que usem *back-channels*, como o FTP (*File Transfer Protocol*). Estes protocolos exigem um pedido de conexão entrando na rede protegida, em resposta a um pedido de conexão saindo, feito por uma máquina desta rede. Por usar regras estáticas, este filtro de pacotes pode ser obrigado a não implementar perfeitamente a política de segurança da rede local para possibilitar estas últimas conexões, permitindo também a entrada de certos pacotes que deveriam ser filtrados no *firewall*.

Surgiram depois filtros mais inteligentes, capazes de criar regras dinamicamente. Estes filtros são conhecidos como filtros de pacotes dinâmicos. Os filtros de pacotes com estados, que serão apresentados na próxima seção, são considerados por muitos somente um avanço de filtros de pacotes dinâmicos (Avolio, 1998) e, por isso, não se tratam de uma nova tecnologia de *firewall*, como falam outros (*CheckPoint*, 1998).

Entretanto, os filtros de pacotes apresentam algumas desvantagens. Aplicações baseadas no protocolo UDP (*User Datagram Protocol*) são difíceis de filtrar, porque no UDP não existe distinção entre uma requisição e uma resposta. Nestes filtros, as soluções utilizadas para tais serviços consiste simplesmente em eliminar as sessões UDP inteiramente, ou ainda em abrir uma porção do espaço de portas UDP para comunicação bi-direcional e, portanto, expor a rede interna. Alguns filtros dinâmicos têm a capacidade de "lembrar" os pacotes UDP que saem da rede interna e, com isso, só aceitam pacotes entrando nesta rede que correspondam a estes primeiros. Esta correspondência é feita mantendo-se algumas informações presentes nos cabeçalhos de rede e de transporte dos pacotes UDP saindo da rede protegida: porta de origem, endereço de origem, porta de destino e endereço de destino. Então, somente pacotes UDP entrando na rede interna, vindos destas portas e endereços do destino original para as suas respectivas portas e endereços da origem, são aceitos. Aplicações baseadas em RPC (*Remote Procedure Call*) são ainda mais complicadas de serem filtradas, por não usarem números de portas pré-definidos. A alocação de portas é feita de maneira dinâmica e geralmente muda com o tempo. O *portmapper/rpcbind* é o único servidor relacionado ao RPC que usa um número de porta pré-definido. Ele é o responsável em manter o mapeamento entre os serviços RPC oferecidos e os números de portas que foram alocados para eles. O que filtros tradicionais geralmente fazem é simplesmente bloquear todos os pacotes UDP, porque a grande maioria destes serviços baseados em RPC utilizam como protocolo de transporte o UDP.

Por excluírem do processo de filtragem os dados da aplicação, tais filtros podem permitir ataques às vulnerabilidades destes protocolos e dos serviços do nível de aplicação. Além disso, filtros de pacotes permitem que máquinas, dentro da rede a ser protegida, recebam diretamente os pacotes enviados pelas máquinas externas, tornando possíveis ataques que se utilizam de vulnerabilidades e falhas no sistema operacional destas máquinas, como por exemplo *teardrop*, *ping of death* e outros.

Como os filtros de pacotes tradicionais não fazem nenhuma filtragem na camada de aplicação, alguma outra tecnologia de *firewall* também deve ser utilizada para eliminar esta deficiência.

2.2 Proxies

Um *proxy* ou *gateway* de aplicação é uma função de *firewall* no qual um serviço de rede é fornecido por um processo que mantém um completo estado da comunicação, examinando todo o fluxo de dados. Um *proxy*, na verdade, evita que *hosts* na rede interna sejam acessíveis de fora da rede protegida, usando duas conexões: uma entre o cliente (máquina interna) e o servidor *proxy* e outra entre o servidor *proxy* e o servidor real (máquina remota).

Um servidor *proxy* pode melhorar a segurança por examinar o fluxo da conexão na camada de aplicação, mantendo informações de contexto para o processo de decisão. Com isso, ele permite fazer uma filtragem na camada de aplicação evitando ataques relacionados às vulnerabilidades dos protocolos de aplicação, possíveis em filtros de pacotes. Além disso, pode ser também utilizado para implementar cache de dados para um determinado serviço e autenticação de usuários. Entretanto, há uma queda de desempenho em relação a filtros de pacotes, porque este processo requer duas conexões e geralmente é implementado como um processo do sistema operacional. Como deve existir um *proxy* para cada serviço diferente, há uma perda na escalabilidade, transformando o suporte a novas aplicações um grande problema. Para amenizar este problema, alguns desses produtos de *firewall* fornecem ferramentas para a criação de *proxies* genéricos, que fazem na realidade, pouco mais que um filtro de pacotes

estático e, portanto, não são muito diferentes de filtros de pacotes dinâmicos.

Um outro problema de muitos produtos que implementam *proxies* é a perda de transparência. Os clientes atrás do *firewall* devem ser configurados para usarem o *proxy* do serviço e não os servidores remotos diretamente. Este problema não existe mais nos *proxies* conhecidos como transparentes. Nestes *proxies* transparentes, os clientes não precisam de nenhuma configuração especial para fazerem a conexão direta ao *proxy*.

Em NAT (*Network Address Translation*) não há consumo de recursos da máquina responsável pela tradução de endereços, por não manter estados de conexões, ao contrário de *proxies* transparentes. Neste caso, há somente uma reescrita nos campos de endereços dos pacotes que entram e saem da rede interna. Para a implementação de um *proxy* transparente é preciso que todo tráfego de pacotes passe ou seja redirecionado para a máquina *proxy*, antes de pesquisar o seu destino. É conveniente, portanto, que o servidor *proxy* fique em uma máquina *dual-homed.* Esta máquina deve aceitar todos os pacotes, saindo da rede interna, vindos dos clientes, independente do endereço do destino. O *proxy* transparente funciona como um *proxy* tradicional, fazendo a comunicação com o servidor remoto. Para o cliente, tudo é transparente porque o servidor *proxy* usa o endereço do servidor remoto como origem dos pacotes que ele envia para o cliente. Portanto, *proxies* transparentes não podem ser confundidos com máquinas que fazem NAT.

Além destes problemas, se um protocolo é muito complexo, a implementação do *proxy* pode ficar com código fonte muito grande e conter *bugs*. Por confiar no sistema operacional (um *proxy* está no espaço de processos do sistema operacional), uma máquina *proxy* pode ter sua segurança comprometida também, por ataques ao sistema operacional. São conhecidos hoje vários problemas em sistemas operacionais que podem entre outras coisas, negar serviços, como por exemplo problemas na remontagem de pacotes com seus fragmentos.

Para alguns serviços pode ser difícil ou até mesmo impossível implementar *proxies*. *Proxies* para serviços baseados em UDP, ICMP (*Internet Control Message Protocol*) e RPC, por exemplo, são muito difíceis de implementar. Além disso, com o crescimento de uso do WWW (*World Wide Web*), vários novos serviços têm aparecido rapidamente, o que torna complexa a tarefa de criar *proxies* para estes novos serviços com a mesma rapidez.

3 FILTRO DE PACOTES COM ESTADOS

Visando aumentar a segurança, um *firewall* deve guardar informações e controlar todo fluxo de comunicação passando através dele. Para fazer decisões de controle para serviços baseados em TCP/IP (aceitar ou rejeitar pedidos de conexões, autenticar, cifrar, etc), um *firewall* deve obter, armazenar, recuperar e manipular informação derivada de todas as camadas de comunicação incluindo a de aplicação.

Não é suficiente examinar pacotes isoladamente. Informações de estados, derivada de comunicações passadas e de outras aplicações, é fator essencial para a decisão de controle em novas tentativas de comunicação. Estas são as idéias básicas de um filtro de pacotes com estados (*CheckPoint*, 1998).

Um filtro com estados potencialmente pode fazer qualquer coisa que um *proxy* pode fazer (Russel, 1997). Mas diferentemente de um *proxy*, ele envolve no seu processo de filtragem todas as camadas de comunicação, desde os cabeçalhos de rede e transporte até a parte de dados do nível de aplicação. Desta forma o processo de filtragem torna-se agora mais homogêneo: uma só entidade vai fazer a análise completa do tráfego de dados. Veremos depois que, embora seja possível fazer o mesmo que quaisquer *proxies*, implementações correntes de filtros com estados não o fazem por completo.

Por manter estados do andamento da comunicação, um filtro de pacotes com estados permite o suporte a protocolos sem conexão, tais como UDP, e a serviços baseados em RPC, que usam portas alocadas dinamicamente. Com estes filtros é possível extrair informações de contexto (o campo de dados) dos pacotes UDP, mantendo desta forma uma conexão virtual de forma a fazer uma filtragem mais inteligente. Para o caso de serviços baseados em RPC, uma cache pode ser mantida nas tabelas de estados do filtro de pacotes com estados, usando temporariamente um processo de realimentação, para um *portmapper/rpcbind* "virtual". Este, por sua vez, pode mapear os números de portas aos serviços RPC oferecidos na rede interna, podendo estabelecer regras dinâmicas para estes serviços. O processo de realimentação contínuo é necessário porque um serviço RPC em algum *host* da rede interna pode, eventualmente, alocar um outro número de porta depois de uma reinicialização do *host*, por exemplo.

No filtro de pacotes com estados, portanto, temos a segurança de um *proxy* e o desempenho e escalabilidade de filtros de pacotes. Deve-se lembrar, contudo, que a escalabilidade não ocorrerá inicialmente com o mesmo grau de segurança que os serviços previamente cobertos. Os níveis de rede e transporte podem receber regras de filtragem apropriadas em questão de minutos, por um administrador de segurança competente. Contudo, o nível de aplicação, para um novo serviço, será geralmente estranho ao administrador e o desenvolvimento de filtragem apropriada poderá levar tempo. Assim, é possível disponibilizar o novo serviço imediatamente, mas em um grau menor de segurança (caso o novo serviço venha realmente exigir filtragem específica no nível de

aplicação) até que regras mais sofisticadas sejam disponibilizadas. Por fazer seu trabalho com todos os *bits* do pacote, tais filtros são implementados nas camadas mais baixas do sistema operacional, geralmente entre as camadas de comunicação de enlace de dados e rede. Isso pode garantir mais segurança por evitar que pacotes "mal formados" cheguem ao sistema operacional da máquina filtradora antes de serem interceptados pelo filtro. Entretanto, ainda permitem, como nos filtros tradicionais, que as máquinas internas sejam acessadas diretamente.

Uma vantagem, que os implementadores de filtros de pacotes com estados alegam que os seus produtos têm em relação a *proxies*, é a de ser mais fácil e rápido o suporte a novos serviços e protocolos. Geralmente estes produtos incluem linguagens de *scripts* para permitir ao usuário do produto criar regras de filtragem para estas novas tecnologias. A verdade é que estas linguagens apenas tornam possível construir mais facilmente *proxies* (podemos pensar que filtros de pacotes com estados implementam *proxies* em filtros de pacotes), melhores que os *proxies* genéricos. *Proxies* genéricos, por não realizarem nenhum tipo de filtragem, devem ser apenas temporários, até que *proxies* dedicados sejam disponibilizados.

Como já foi mencionado antes, um filtro de pacotes com estados pode potencialmente fazer qualquer coisa que um *proxy* faz, ou seja, ele pode implementar um *proxy* em suas regras de filtragem. Entretanto, a realidade, que encontramos nos diversos produtos que implementam filtros de pacotes com estados, é que os mesmos não o fazem por completo. Usaremos o *Firewall-1* (*CheckPoint*, 1997) como exemplo, embora o mesmo possa ser observado em outros produtos de filtros de pacotes com estados. No *Firewall-1*, existem dois módulos básicos: o *Firewall-1 Inspection Module* (também chamado *INSPECT engine*) e o *security server*. No primeiro, reside o filtro de pacotes com estados propriamente dito. Ele é implementado no *kernel* do sistema operacional entre as camadas de enlace de dados e rede. Com o *INSPECT engine* é possível filtrar pacotes usando todas as camadas de comunicação e estados de comunicações passadas. Muito do que pode ser feito em *proxies* pode ser feito também neste módulo, porque ele implementa, com suas regras de filtragem, *proxies* mais simples para os serviços. Entretanto, se o usuário quiser fazer um trabalho maior na camada de aplicação, ele precisa usar o segundo módulo do produto. Os *security servers* rodam no espaço de processos e existem para FTP, HTTP (*HyperText Transmission Protocol*) , SMTP (*Simple Mail Transfer Protocol*) e também para fins de autenticação. Na verdade, eles apresentam as mesmas características que *proxies* para estes serviços. Assim, quando o usuário desejar autenticação extra de usuários, fazer inspeção no fluxo de dados à procura de vírus, bloquear *applets Java* ou fazer análises mais sofisticadas no fluxo de dados, o *Firewall-1* deixa estes *security servers* realizarem o trabalho. A vantagem desta solução é que o filtro no *kernel* fica mais simples, eliminando o *overhead* de um *proxy* implementado junto do filtro e garantindo, portanto, um bom desempenho com segurança, além de regras de filtragens mais simples, o que diminui as chances de erros.

Na prática, portanto, a maioria dos produtos que implementam filtros de pacotes com estados recorrem a *proxies* (os *security servers*, no caso particular do *Firewall-1*) sempre que é necessário fazer um trabalho maior com os dados da aplicação. Por este motivo, muitos acreditam que *proxies* são mais seguros que filtros de pacotes com estados. Além disso, suspeita-se que os produtos comerciais que implementam filtros de pacotes com estados, sequer fazem uma análise mais detalhada no fluxo de dados para alguns serviços e protocolos, funcionando como um simples filtro de pacotes tradicional (Avolio, 1998). Na próxima seção voltaremos a esta discussão.

Finalizando o tema sobre "filtros de pacotes com estados", falta mencionar que a implementação de NAT no mesmo produto, junto com filtros de pacotes com estados, é bastante facilitada; por isso, a maioria dos produtos desta classe oferece este recurso. NAT, para a maioria dos protocolos e serviços, deve trabalhar no nível de rede, mas para alguns outros, ele deve fazer algum trabalho também na camada de aplicação. Este é o caso do FTP (se houver tradução de endereços somente no nível de rede, o FTP não funcionará, uma vez que ele usa endereços de rede em seu processamento). Deste modo, NAT assim como filtros de pacotes com estados, também realiza seu trabalho nas camadas de rede e aplicação e pode ser implementado usando-se os recursos da aplicação de filtragem, sem grande esforço adicional. Além disso, usando NAT e as informações de estados, filtros de pacotes com estados podem oferecer suporte à distribuição de processamento entre diversos servidores de algum serviço de maneira transparente, dividindo o carregamento entre os diversos servidores e provendo escalabilidade e maximizando o desempenho. Isso é muito útil, principalmente em redes locais que permitem acesso a suas informações através de sites *Web*, onde é grande o número de acessos. Esta característica também pode ser encontrada na maioria dos produtos que implementam filtros de pacotes com estados.

4 FILTROS DE PACOTES COM ESTADOS VS. PROXIES

Como visto na seção anterior, um filtro de pacotes com estados geralmente pode implementar um *proxy* mais simples no *kernel* do sistema operacional e, ao mesmo tempo, funcionar como

um filtro de pacotes tradicional. Além disso, alguns participantes em listas de discussão da área acreditam que, para alguns protocolos e serviços, os produtos que implementam filtros de pacotes com estados não fazem nenhuma análise no fluxo de dados, funcionando como um filtro de pacotes tradicional (Avolio, 1998). Mas será que por isso é sempre verdade falar que *proxies* são soluções melhores que filtros de pacotes com estados? A resposta para esta pergunta é não. Dependendo de que requisitos devem ser satisfeitos em uma rede local, filtros de pacotes com estados, sozinhos, podem ser suficientes para prover segurança.

Teoricamente, é possível construir um *proxy* completo em um filtro de pacotes com estados (Russel, 1997). Desta forma, poder-se-ia afirmar que um *proxy* é um caso especial de um filtro de pacotes com estados. Entretanto na prática como vimos anteriormente, os vendedores optam por produtos mais leves e, quando um trabalho maior precisa ser feito na camada de aplicação, eles recorrem a *proxies* para o serviço. Isso pode ser explicado pelo fato de que é uma tarefa mais complexa escrever regras de filtragem que implementem um *proxy* do que escrever um *proxy* equivalente. Mas, se o ambiente computacional não exige, em nenhum serviço ou protocolo, um trabalho maior na camada de aplicação, um filtro de pacotes com estados é uma solução suficiente. Se este não é o caso, pode ser necessário usar um *proxy* para o serviço junto com o filtro de pacotes com estados ou com um filtro de pacotes tradicional.

O filtro de pacotes com estados também tem a vantagem de evitar ataques às vulnerabilidades do sistema operacional, por interceptar os pacotes antes de atingirem o sistema operacional, evitando assim os diversos ataques de baixo nível na arquitetura TCP/IP, como por exemplo, *stealth scanning*, SYN *flood*, etc. Além disso, serviços baseados em UDP e RPC podem ser tratados convenientemente neste tipo de filtros de pacotes, como foi discutido na seção anterior. O NFS (*Network File System*), por exemplo, é um serviço baseado em RPC e UDP que poderia ser mais facilmente provido por um filtro de pacotes com estados. Construir um *proxy* para NFS é ainda mais complicado que em outros serviços baseados em UDP e RPC, porque neste serviço o desempenho é fundamental e o volume de dados é grande.

É, portanto, inegável que filtros de pacotes com estados podem ser uma solução suficiente em diversos ambientes computacionais onde os riscos são baixos e em ambientes com exigências de desempenho. Se algum trabalho mais sofisticado deve ser feito como, por exemplo, bloquear instâncias específicas de *Java* ou buscar vírus no fluxo de dados, deve-se usar *proxies* dedicados a este trabalho, em adição à filtragem por estados. No caso de alguns produtos comerciais como o *Firewall-1*, vimos que é possível que estes *proxies* já sejam incluídos no próprio produto. Esta necessidade pode ocorrer, na verdade, com serviços novos ou com serviços e protocolos mais complexos, onde *proxies* mais poderosos sejam a única maneira de garantir segurança ao serviço. Como exemplos, cita-se o caso de clientes WWW e o de propagação de vírus e cavalos de Tróia (*BackOriffice*, *Netbus* e semelhantes).

No caso do suporte a serviços e protocolos novos, as linguagens de *scripts* nestes produtos comerciais que implementam filtros de pacotes com estados, para a escrita de regras de filtragem, podem permitir uma solução mais segura que *proxies* genéricos ou neutros. Além disso, para aplicações desenvolvidas dentro de um ambiente corporativo, por exemplo, para as quais não existem *proxies*, pode ser bastante interessante usar um filtro de pacotes com estados para estabelecer regras de filtragem para este serviço.

É válido ressaltar ainda que, por seu baixo custo computacional e por sua segurança adicional, filtros de pacotes com estados são sempre preferidos a filtros de pacotes estáticos e dinâmicos.

Em resumo: o filtro de pacotes com estados sempre é uma boa solução, mesmo que seja apenas em substituição aos filtros tradicionais em ambientes onde os riscos são altos. *Proxies*, sempre em adição à filtragem, serão acrescentados sempre que algum tratamento mais sofisticado precisar ser realizado em algum serviço. No âmbito de *firewalls*, já existem vários produtos na categoria de filtros de pacotes com estados. *Firewall-1* da *CheckPoint*, *CISCO-PIX* da *Cisco*, o *Firewall/Plus* da *Network-1*, o *Guardian* da *NetGuard* e o *Gaunlet* da *TIS*, são alguns dos que implementam filtros de pacotes com estados.

5 LIMITAÇÕES DAS TECNOLOGIAS DE FIREWALL

Todas estas tecnologias de *firewall* infelizmente ainda apresentam algumas limitações (Pouw e Geus, 1997). Dentre as limitações, o surgimento destas linguagens de *scripts*, como *Java* e *Javascript*, que fazem a *Web* tão atrativa, são um grande problema para os *firewalls*. Bloquear tais programas não é a melhor solução, mas é a única que há no momento para os projetistas de *firewalls*. Seria interessante algo que permitisse uma filtragem seletiva destes programas no firewall. Infelizmente, isso é impossível em alguns casos. Talvez uma solução distribuída (no *firewall* e nos clientes atrás do *firewall*, por exemplo) resolva este problema, mas até o momento as soluções existentes não são 100% seguras.

Um outro grande problema para *firewalls* é uma classe de ataques que usa o "tunelamento" de pacotes, isto é, dados não permitidos dentro de pacotes permitidos, segundo a análise do *firewall*. Já são conhecidos ataques que usam esta deficiência

dos protocolos TCP/IP. Como um exemplo, vamos imaginar um programa "cavalo de Tróia" que consegue privilégios de superusuário dentro de uma rede local. O atacante pode simplesmente aguardar que este programa atrás do *firewall* tente estabelecer uma conexão para um servidor do atacante e este servidor, na verdade, envia para ele os comandos a serem executados dentro da rede local. Para que os pacotes passem desapercebidos no *firewall*, o programa atacante atrás do *firewall* e o servidor do atacante podem simplesmente encapsular seus dados dentro de pacotes HTTP que seriam permitidos pelo *firewall*. Não existe nenhuma tecnologia de *firewall* que possa evitar este tipo de ataque. Algo que poderia ser feito seria exigir que as tentativas de conexões saindo da rede fossem autenticadas, mas se aquele programa sendo executado atrás do *firewall* for um pouco mais sofisticado, esta solução não será suficiente.

Portanto, é importante que mesmo com um *firewall*, defina-se políticas de segurança para os usuários da rede local (evitar a execução de programas recebidos por *e-mail* e de *applets Java* em páginas WWW não confiáveis, por exemplo) e usar outras ferramentas de segurança (IDS, anti-vírus atualizados, entre outras) minimizando estas limitações das tecnologias de *firewalls*.

6 CONCLUSÃO

Filtros de pacotes com estados podem ser considerados um avanço ou uma nova geração de filtros de pacotes; em alguns ambientes podem ser suficientes para prover segurança. Estes ambientes são ambientes onde os riscos são baixos ou deseja-se eliminar os gargalos de comunicação via *proxy*, provendo transparência aos usuários e garantindo segurança maior que filtragem simples.

Se o ambiente computacional quer minimizar riscos de segurança, algum trabalho extra deve ser feito para alguns serviços isolados, usando-se então *proxies* dedicados e bastante focados.

Estas tecnologias de *firewall* não conseguem, por si só, garantir um ambiente totalmente seguro. Políticas de segurança para os usuários devem ser bem estabelecidas para garantir que certos problemas explorando limitações dos *firewalls* sejam evitados.

AGRADECIMENTOS

Os autores agradecem ao CNPq, à FAEP-UNICAMP e ao Pronex-SAI pelo financiamento ao trabalho e às muitas pessoas das listas de discussão de segurança, que direta e indiretamente nos ajudaram com referências e informações sobre o assunto.

REFERÊNCIAS BIBLIOGRÁFICAS

AVOLIO, Frederick M. *Application Gateways and Stateful Inspection: Brief Note Comparing and Contrasting.* Trusted Information System, Inc. (TIS). Janeiro, 1998.

CHAPMAN, D. Brent; ZWICKY, D. Elizabeth. *Building Internet Firewalls*. O' Reilly & Associates, Inc. 1995.

CHECKPOINT, Software Technologies Ltd. *CheckPoint Firewall-1 White Paper, Version 3.0*, junho, 1997.

CHECKPOINT, Software Technologies Ltd. *Stateful Inspection Firewall Technology*, Tech Note, 1998.

CHESWICK, William ; BELLOVIN, Steven M. *Firewalls and Internet Security – Repelling the wily hacker*. Addison Wesley, 1994.

GONÇALVES, Marcus. *Firewalls Complete*. McGraw-Hill, 1997.

POUW, Keesje Duarte. *Segurança na Arquitetura TCP/IP: de Firewalls a Canais Seguros.* Campinas: IC-Unicamp, Janeiro, 1999. Tese de Mestrado.

POUW, Keesje Duarte; GEUS, Paulo Licio de. *Uma Análise das Vulnerabilidades dos Firewalls.* WAIS'96 (Workshop sobre Administração e Integração de Sistemas), Fortaleza. Maio, 1997.

RUSSEL, Ryan. *Proxies vs. Stateful Packet Filters.* http://futon.sfsu.edu/~rrussell/spfvprox.htm, Junho 1997.

SPONPH, Marco Aurélio. *Desenvolvimento e análise de desempenho de um "Packet Session Filter"*. Porto Alegre – RS: CPGCC/UFRGS, 1997. Tese de Mestrado.