FILTRAGEM COM ESTADOS DE SERVIÇOS BASEADOS EM RPC NO KERNEL DO LINUX

RESUMO

As tecnologias mais tradicionais de firewall – filtros de pacotes e agentes proxy – apresentam limitações que dificultam o suporte a certos protocolos e serviços TCP/IP, principalmente aqueles serviços que usam portas alocadas dinamicamente. Filtros de pacotes com estados são uma boa alternativa, pois permitem a criação de regras de filtragem dinâmicas para tais serviços. Este trabalho mostra detalhes da implementação de um filtro de pacotes com estados capaz de tratar serviços baseados em RPC. Foi usado como ponto de partida o filtro com estados implementado no novo kernel do Linux (série 2.4). O resultado deste trabalho foi aprovado e será incorporado à próxima versão oficial do núcleo deste sistema operacional.

ABSTRACT

Most traditional firewall technologies, such as packet filters and proxy agents, present limitations that make it difficult to support some TCP/IP protocols and services, mainly those services that use ports dynamically allocated. Stateful Packet Filter are a good alternative, because permits the setup of dynamic filtering rules for such services. This work shows details of the implementation of a stateful packet filter able to handle RPC-based services. It was used like start point the stateful filter implemented in new Linux kernel (2.4 series). The result of this work was approved and will be officially incorporated into the next operating system version.

1 INTRODUÇÃO

Existem diversos serviços e protocolos de aplicação que podem ser responsabilizados pelo crescimento surpreendente da Internet. Muitos outros estão surgindo rapidamente sobre a infra-estrutura de rede atual como resultado deste sucesso, impulsionados pela tecnologia e devido à pressão de usuários cada vez mais exigentes. Além disso, o aumento na velocidade das linhas de comunicação e no desempenho das máquinas têm facilitado o desenvolvimento de protocolos mais complexos. Tais protocolos envolvem a movimentação de uma grande diversidade de dados pela rede, em quantidades variáveis, com exigências diferentes na qualidade de serviço e, em geral, usam várias portas alocadas dinamicamente para as diferentes sessões ou conexões necessárias.

Por outro lado, a maioria destes serviços não trazem muitos mecanismos de segurança embutidos e, geralmente, herdam muitos dos problemas da suíte TCP/IP original. É importante, portanto, que as tecnologias de segurança sejam capazes de agregar segurança a estes serviços para que usuários, em redes locais com maiores requisitos de segurança, possam usufruir destes atraentes serviços e protocolos. Infelizmente, as tecnologias tradicionais de *firewall* têm suas limitações e nem sempre conseguem tratar convenientemente todos estes interessantes serviços existentes na rede. Serviços baseados em UDP e RPC são representantes importantes desta classe de serviços. Neste trabalho serão tratados exclusivamente os problemas relacionados a serviços baseados em RPC. O leitor mais interessado em uma discussão mais geral sobre o assunto pode buscar informações na referência [12].

Na seção 2, serão discutidos os problemas existentes para a filtragem de serviços baseados em RPC. A seção 3 apresenta brevemente a nova geração de filtros de pacotes – filtros de pacotes com estados (*Stateful Packet Filter, SPF*) – e como filtros com estados podem solucionar os problemas mencionados na seção 2. A seção 4 mostra detalhes da implementação de um SPF no *kernel* de desenvolvimento 2.3.x do Linux. Na seção 5 será visto como este filtro da seção 4 foi estendido para oferecer suporte a serviços baseados em RPC. Finalmente, a seção 6 traz algumas considerações finais e a conclusão do trabalho.

2 SERVIÇOS BASEADOS EM RPC

Existem vários protocolos de chamada a procedimentos remotos conhecidos como RPC. O mais popular e que será usado em nossa discussão, neste trabalho, é o Sun RPC, que foi originalmente desenvolvido pela Sun Microsystems [15].

Nos protocolos UDP e TCP, os cabeçalhos dos pacotes reservam somente 2 *bytes* para os campos com números de portas, ou seja, existem somente 65.536 possíveis portas para todos os serviços TCP e UDP (65.536 para cada). Se tivesse que reservar um número de porta bem definido para cada serviço existente, ter-se-ia um número de serviços limitado por este valor. Entre outras coisas, RPC oferece uma boa solução para este problema [2]. Cada serviço baseado em RPC recebe um número de programa único de quatro *bytes* (isto permite

4.294.967.296 serviços diferentes). Como RPC é implementado acima do nível de transporte, deve existir algum mecanismo de mapeamento entre os números dos serviços RPC, oferecidos em uma máquina, e os números de porta que estes serviços estão usando em um dado momento. No Sun RPC, o Portmapper/Rpcbind é o responsável por este mapeamento e é o único servidor baseado em RPC que usa um número de porta pré-definido (porta 111, em ambos TCP e UDP).

Quando um servidor RPC é iniciado, ele aloca um número de porta qualquer e registra-se junto ao Portmapper/Rpcbind. Nesta comunicação, ele passa o seu número de programa, a porta usada, o número de versão e o número do protocolo para o Portmapper/Rpcbind a ser usado na comunicação com os clientes. Vale comentar que o servidor usa o próprio protocolo RPC nesta comunicação. Estas informações são, na verdade, os parâmetros para um procedimento PMAPPROC_SET a ser executado pelo Portmapper/Rpcbind. Um cliente, desejando comunicar-se com este servidor RPC, comunica-se com o Portmapper/Rpcbind para consultar o número de porta usado pelo serviço. Na verdade, ele também faz uma chamada remota ao procedimento PROC_GETPORT, passando como parâmetros o número de programa, a versão e o protocolo usado. No momento em que o cliente obtém a porta usada pelo servidor, ele pode comunicar-se diretamente com o serviço RPC, sem nenhum envolvimento do Portmapper/Rpcbind.

Por não usarem números de portas pré-definidos, não se tem como criar regras estáticas para filtragem de serviços baseados em RPC. A alocação de portas é feita dinamicamente e pode mudar com o passar do tempo (máquina ou servidor podem ser reinicializados, por exemplo). Bloquear tráfego de pacotes para o Portmapper/Rpcbind não resolve, pois um atacante pode descobrir o número da porta usada por algum serviço baseado em RPC, fazendo tentativas para todas as portas não reservadas. Outros serviços RPC podem usar números de portas pré-definidos, como é o caso do NFS que usa a porta 2049 [14]. Mesmo assim, é aconselhável bloquear acesso ao Portmapper/Rpcbind para evitar que eles sejam usados para ataques à própria máquina. O Portmapper/Rpcbind tem um recurso chamado *proxy forwarder*, que permite que clientes façam uma chamada ao procedimento PMAPPROC_CALLIT, passando como parâmetro o número de um programa RPC, sua versão, um número de procedimento deste serviço e os parâmetros para este procedimento. O Portmapper/Rpcbind faz a chamada ao procedimento do serviço, caso esteja registrado, e retorna os resultados obtidos para o cliente. Notar que isto pode ser perigoso para alguns serviços, uma vez que o Portmapper/Rpcbind tem privilégios de super-usuário e a requisição parece vir de uma máquina confiável (a própria máquina).

A solução encontrada para a filtragem de serviços baseados em RPC, com filtros de pacotes estáticos, é simplesmente bloquear todo tráfego de pacotes UDP, pois a grande maioria destes serviços usam UDP como protocolo de transporte [2].

3 FILTROS DE PACOTES COM ESTADOS (*STATEFUL PACKET FILTER* - SPF)

Filtros de pacotes tradicionais não mantêm informações de estados das conexões ou sessões que passam pelo *firewall* [12]. Pacotes são analisados independentemente, usando somente as informações existentes nos seus cabeçalhos de rede e transporte e regras de filtragem. Se um pacote qualquer com as *flags* de sincronismo SYN e ACK ligadas chegar ao filtro de pacotes, o filtro supõe que este seja parte de algum canal virtual existente entre a rede interna e a rede externa. Silenciosamente, este o aceita, caso esteja de acordo com o conjunto de regras que implementa a política de segurança local.

Entretanto, um bom *firewall* não deve permitir que pacotes não válidos entrem na rede a ser protegida. Pacotes não válidos são aqueles que não satisfazem as regras de filtragem e/ou não pertencem a nenhuma conexão ou sessão existente através do *firewall*. Isto, além de garantir a perfeita implementação da política de segurança da rede interna, evita diversos ataques já bem conhecidos e outros que poderão aparecer no futuro usando pacotes devidamente construídos e/ou mal formados. Um bom exemplo destes ataques são os pacotes ou fragmentos de pacotes, que satisfazem às regras de filtragem, mas não pertencem a nenhuma conexão ou sessão existente entre a rede interna e a externa, enviados para derrubar uma máquina alvo, explorando alguma falha na implementação da pilha de protocolos TCP/IP de um determinado sistema operacional existente na rede interna. Aliás, um atacante pode descobrir os sistemas operacionais usados em máquinas internas, enviando alguma seqüência bem definida de pacotes com características não especificadas nos padrões dos protocolos e, de acordo com as respostas obtidas, inferir sobre o sistema usado pela máquina.

Em um filtro de pacotes com estados, quando um pacote SYN/ACK chega ao filtro, ele normalmente verifica, em suas tabelas de estados das conexões e sessões, se este pertence a alguma conexão em andamento através do *firewall*. Caso não exista uma conexão TCP válida para este pacote, ele é descartado e, possivelmente, registrado nos *logs*. Portanto, mesmo que o filtro esteja configurado com uma única regra

permitindo todo tráfego de pacotes entre as redes, estes pacotes continuarão sendo bloqueados no *firewall*, já que não pertencem a nenhuma conexão válida segundo suas tabelas de estados. Esta solução garante sempre uma maior segurança, visto que somente pacotes autorizados pelas regras de filtragem ou pertencentes às sessões e conexões válidas entre as redes podem atravessar o filtro. Uma outra vantagem desta solução é a melhora no desempenho, pois somente os primeiros pacotes das conexões ou sessões são verificados com as regras de filtragem, que geralmente é uma tarefa de mais alto custo computacional.

Vale ressaltar aqui que este é o comportamento mais normal das atuais implementações de filtros de pacotes com estados. Nem sempre este processo de filtragem com estados trabalha exatamente desta forma em todas as implementações. Portanto, não existe uma especificação padrão para a implementação de um SPF, ou seja, o termo "*stateful*" especifica que o filtro deve manter informações de estados das sessões e conexões, mas não como estas informações devem ser mantidas nem como elas devem ser usadas no processo de filtragem.

Para alguns serviços e protocolos de aplicação, como o caso do FTP, algum trabalho extra precisa ser feito também no nível de aplicação. Estes codificam certas informações dos níveis de rede e transporte, como números de portas e endereços IP, na parte de dados dos pacotes. Para estes casos especiais, filtros de pacotes com estados podem acompanhar o fluxo de dados da aplicação, extrair as informações necessárias para suas tabelas de estados e para criação de regras dinâmicas. No caso do FTP, o filtro pode extrair, por exemplo, a porta que o cliente usará para uma conexão de dados. Desta forma, é possível garantir o correto funcionamento destes protocolos e serviços.

Por já fazerem algum trabalho nesta camada de comunicação para alguns serviços, alguns SPFs, como o CISCO-PIX e o FW-1, permitem também alguma filtragem no nível de aplicação, como em agentes *proxy*. Entretanto, a filtragem neste caso é geralmente bastante simples, verificando somente a parte de dados dos pacotes individualmente, sem qualquer remontagem do fluxo de dados do serviço.

Apesar disso, muitos serviços de aplicação transmitem comandos do protocolo e suas respectivas respostas em unidades pequenas, que cabem em um único datagrama IP e podem perfeitamente ser filtrados por filtros de pacotes com estados. Um bom exemplo é o caso da comunicação entre clientes RPC e o Portmapper, que será destacada neste trabalho. As mensagens enviadas pelos clientes, requisitando o número da porta usada por algum servidor RPC, ocupam somente 56 *bytes*, enquanto uma resposta com o número da porta usada ocupa 28 *bytes* da parte de dados. Nestes casos, não há a necessidade de remontagem dos dados dos pacotes de uma comunicação para filtrar comandos do serviço ou protocolo de aplicação. A grande vantagem obtida com esta filtragem é que o SPF desempenha o papel de um agente *proxy* dedicado sem comprometer o desempenho.

Por manter estados das conexões e sessões, um filtro de pacotes com estados permite também associar pacotes ICMP de erro ou controle à sua devida conexão ou sessão. Estes pacotes trazem informações que permitem identificar a conexão ou sessão da máquina à que se refere a mensagem ICMP. Usando estas informações, o filtro pode verificar a existência desta sessão ou conexão em suas tabelas de estados e permitir a passagem dos mesmos.

3.1 Filtragem com estados de serviços baseados em RPC

Foi visto na seção 2 que a característica de não usar números de porta pré-definidos é o principal problema para a filtragem estática de serviços baseados em RPC. Um servidor RPC aloca o número de porta dinamicamente, junto ao sistema operacional, e registra-se ao servidor de nomes portmapper/rpcbind, informando a porta sendo usada em um determinado momento. Isto impossibilita a criação de regras estáticas para estes serviços usando número de portas fixos e previamente conhecidos.

Uma boa solução para este problema é utilizar informações de estados extraídas da comunicação entre um cliente RPC e o portmapper/rpcbind para possibilitar a criação de regras dinâmicas para tais serviços. Um filtro de pacotes com estados pode simplesmente acompanhar uma sessão UDP (ou uma conexão TCP) envolvendo clientes RPC e o portmapper/rpcbind, dando atenção especial aos dados trocados em todas as chamadas ao procedimento PMAPPROC_GETPORT. Associando requisições às suas respostas e extraindo as informações relevantes, é possível manter as informações das portas sendo usadas e, consequentemente, permitir a comunicação entre clientes e servidores RPC.

Esta solução de acompanhar todas as sessões entre clientes RPC e o portmapper/rpcbind e extrair as informações necessárias foi implementada neste trabalho, como será visto mais adiante na seção 5.

4 – IPTABLES: UM FILTRO DE PACOTES COM ESTADOS NO KERNEL DO LINUX

Com o aumento no desempenho dos processadores, alguns sistemas operacionais

passaram a implementar o processo de filtragem de pacotes no núcleo do sistema. Nas versões 2.0.x e 2.2.x de produção do *kernel* do Linux, por exemplo, pode-se encontrar respectivamente os filtros Ipfwadm e Ipchains, que implementam simplesmente uma filtragem estática dos pacotes. Neste trabalho, entretanto, tem-se mais interesse na nova infra-estrutura usada para tratamento e filtragem dos pacotes que estará presente na série de produção 2.4 do *kernel*, neste sistema operacional.

Como ainda não há uma versão estável deste novo *kernel*, muito do que será apresentado neste documento é baseado nas versões 2.3.x de desenvolvimento e poderá sofrer pequenas mudanças até a sua versão final. Contudo, as idéias básicas usadas na implementação certamente permanecerão as mesmas.

4.1 Netfilter

O Netfilter é um *framework* independente da interface normal de Berkeley *Sockets*, que gerencia e organiza como os pacotes que chegam ao *kernel* do Linux devem ser tratados, nas séries de desenvolvimento 2.3 e de produção 2.4. Ele define, por exemplo, que partes do *kernel* podem realizar algum trabalho com o pacote quando este atingir certos pontos do núcleo do sistema operacional [13].

Nesta nova infra-estrutura, cada protocolo define "*hooks*" (IPv4 define 5), que são pontos bem definidos na pilha de protocolos por onde passam os pacotes. Quando um pacote chega a um determinado *hook*, o Netfilter checa que módulos do *kernel* estão registrados para oferecer algum tratamento especial naquele ponto. Caso existam, cada módulo seqüencialmente recebe o pacote (desde que, obviamente, ele não seja descartado ou passado para o espaço de processos por um módulo anterior) e pode, portanto, examiná-lo, alterá-lo, descartá-lo, injetá-lo de volta ao *kernel* ou passá-lo para o espaço de processos, dependendo da funcionalidade de rede que esteja implementando. Estes módulos geralmente registram-se em mais de um *hook* ao mesmo tempo, podendo, até mesmo, fazer trabalhos diferentes com os pacotes em cada um deles.

Desta forma, pode-se definir quais módulos devem ser carregados e registrados perante o Netfilter, sempre que algum tratamento especial com os pacotes for necessário. Além disso, esta interessante infra-estrutura facilita a implementação de novos módulos e permite que os módulos pré-existentes sejam facilmente estendidos, usando interfaces de comunicação bem definidas. Alguns dos módulos nativos são o NAT, o Iptables e o módulo de *Connection tracking*.

Notar que, diferentemente das versões mais antigas do *kernel* do Linux, as diversas funções de rede são implementadas em módulos independentes. Nas versões 2.0.x e 2.2.x, o IP *masquerading*, por exemplo, era implementado junto ao processo de filtragem de pacotes e das funções de roteamento no *kernel*. Isso tornava o código confuso e acarretava alguma perda no desempenho. Com este novo esquema, IP *masquerading*, por ser um caso especial de NAT, é implementado usando o módulo de NAT. Este último, por sua vez, é totalmente independente do módulo de filtragem.

4.2 Módulo de Connection Tracking (Conntrack)

Este módulo foi originalmente implementado para ser usado pelo módulo de NAT. Todavia, possibilita também que os outros módulos no *kernel* possam utilizá-lo. O papel do módulo Conntrack é manter informações de estados de todas as conexões e sessões. O módulo Iptables, em suas versões iniciais, não fazia filtragem com estados, mas foi rapidamente e facilmente estendido para basear seu processo de decisão também nas informações de estados mantidas pelo módulo Conntrack. Portanto, a grosso modo, este módulo de *Connection tracking* é o verdadeiro responsável pela parte "*stateful*" do filtro de pacotes implementado no *kernel* do Linux.

Cada pacote chegando ao kernel é associado a uma estrutura de dados (skb_buff). Todos os componentes do núcleo do sistema operacional, que realizam algum trabalho com um determinado pacote, recebem o apontador para a estrutura de dados correspondente. Desta forma, os cabeçalhos e dados do pacote não precisam ser recopiados durante sua travessia pela pilha de protocolos. Portanto, esta estrutura de dados, dentre outras coisas, tem apontadores para os cabeçalhos dos diferentes níveis de rede e para a parte de dados do pacote. Nas séries 2.3 e 2.4, ela apresenta um campo extra (nfmark), no qual o módulo Conntrack codifica o estado para o pacote em sua sessão ou conexão. Vale notar que este módulo não deve descartar pacotes, com exceção de alguns casos excepcionais (pacotes mal formados). Esta função é deixada para o módulo de filtragem (Iptables). Portanto, depois de passar pelo módulo Conntrack, um pacote atravessa o *kernel* com o seu respectivo estado, que pode ser:

INVALID: para pacotes que não pertencem e não criam uma conexão ou sessão. Pacotes ICMP não válidos são um bom exemplo;

NEW: para pacotes que iniciam uma nova conexão ou sessão;

ESTABLISHED: para pacotes pertencendo a uma conexão já estabelecida;

RELATED: para pacotes relacionados a alguma conexão ou sessão já estabelecida. Por exemplo, pacotes ICMP relacionados a alguma conexão existente.

O módulo de *Connection tracking* atribui um estado a cada pacote que chega ao *kernel*, de acordo com o andamento da comunicação a que pertence. Para isso, ele usa uma tabela *hash* (conhecida como *Separate Chaining*) para manter as informações de estados de todas as conexões e sessões passando pelo núcleo do sistema operacional. Nesta tabela, cada posição guarda uma lista encadeada, onde cada nó representa uma conexão ou sessão (estrutura ip_conntrack_tuple_hash), contendo as informações necessárias para identificá-la unicamente (os endereços IP de origem e destino, os números de portas de origem e destino, o protocolo usado), apontadores para os seus vizinhos na lista e para uma outra estrutura de dados (ip_conntrack) com informações de estados propriamente ditas referentes à comunicação. Usando estas informações, o módulo de Contrack consegue atribuir o estado adequado para cada pacote dentro de sua respectiva conexão ou sessão.

Além da tabela de estados das conexões e sessões, o módulo Conntrack também mantém uma lista de conexões ou sessões aguardadas (expect_list). Cada nó nesta lista guarda as informações dos endereços IP de origem e destino envolvidos, o número da porta destino e o protocolo a ser usado. Se um pacote tiver as mesmas informações em seus cabeçalhos de rede e transporte de uma das entradas nesta lista, o módulo assume que este pertence a esta conexão ou sessão esperada e atribui o estado *RELATED* ao pacote. Dai em diante, todos os pacotes desta conexão ou sessão relacionada recebem sempre o estado de *RELATED*. Além disso, cria-se uma entrada na tabela de estados para esta nova conexão ou sessão e retira a entrada correspondente da lista de conexões ou sessões aguardadas. Isto é equivalente a criar uma regra dinâmica para permitir a comunicação.

Esta lista é, portanto, essencial para permitir o funcionamento correto de certos serviços e protocolos TCP/IP. O FTP é um bom exemplo destes protocolos, pois define o número de porta a ser usado na conexão de dados durante a conexão de controle [14]. Para acomodar bem tais serviços e protocolos, o módulo Conntrack oferece módulos auxiliares para alguns protocolos (conhecidos como "*helpers*") e permite que outros, para protocolos ainda não suportados, sejam facilmente implementados e agrupados ao módulo de *Connection tracking*. Os *helpers* acompanham os dados trocados em uma comunicação envolvendo algum determinado protocolo e extraem as informações necessárias para permitir as conexões ou sessões relacionadas. Com estas informações, este módulo auxiliar pode criar uma entrada para a conexão ou sessão na lista.

Cada uma destas entradas permanece nesta lista enquanto durar a conexão ou sessão que a originou ou até que o primeiro pacote da sessão ou conexão relacionada chegue. Isto significa que não há *timeouts* associados, podendo dificultar um pouco o suporte a certos protocolos. Pode ser necessário, dependendo do protocolo, que um determinado *helper* gerencie uma lista de sessões ou conexões aguardadas particular para o correto tratamento de conexões ou sessões relacionadas. Esta foi a solução adotada para o caso do RPC que será visto na próxima seção.

4.3 Iptables

O Iptables [13] é considerado um filtro de pacotes com estados porque ele permite o uso, no seu processo de filtragem, do estado atribuído pelo módulo de *Connection tracking* aos pacotes das diferentes sessões ou conexões. O módulo Iptables sozinho implementa somente uma filtragem estática dos pacotes como seus antecessores: Ipfwadm e Ipchains. Entretanto, quando o módulo Conntrack está presente no *kernel*, mantendo os estados das conexões ou sessões, o Iptables permite que o administrador de segurança crie regras baseadas nos estados dos pacotes dentro de suas respectivas conexões ou sessões. Desta forma, é possível a criação de regras permitindo a passagem, através do *firewall*, de todos os pacotes com estados *ESTABLISHED* e *RELATED*, sem comprometimento da segurança requerida.

5 – ESTENDENDO O IPTABLES

5.1 Mensagens RPC

O protocolo RPC [14] envolve basicamente dois tipos de mensagens: mensagens "*call*" e mensagens "*reply*". Uma mensagem *call* é originada quando um cliente RPC envia uma requisição para a execução de um procedimento em particular. Depois que o procedimento é executado, o servidor RPC envia de volta uma mensagem *reply*, contendo os resultados retornados na execução do procedimento. A Figura 1 mostra o formato de uma mensagem *call*, quando encapsulada dentro de um datagrama UDP.

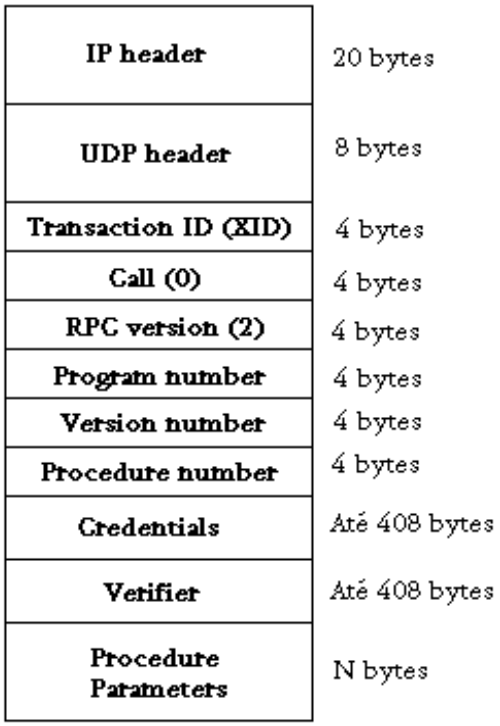

Figura 1: Mensagem Call com UDP

A mensagem *call* inicia com o campo XID (*Transaction* ID) que é um valor inteiro estabelecido pelo cliente e retornado sem nenhuma alteração pelo servidor. Quando o cliente recebe uma mensagem *reply* do servidor, ele compara o campo XID desta última mensagem (observar a existência do campo também na Figura 2) com o XID enviado. Desta forma, o cliente RPC pode verificar se a mensagem recebida se trata de uma resposta válida à sua requisição. É interessante ressaltar ainda que, caso o cliente retransmita a sua mensagem *call*, o valor usado anteriormente no campo XID, na mensagem original, não é alterado. Isto evita problemas com mensagens duplicadas ou atrasadas na rede.

O campo seguinte serve para distinguir entre mensagens *call* (valor inteiro 0) e mensagens *reply* (valor inteiro 1). A versão corrente do protocolo Sun RPC é 2 e, portanto, é este valor que é sempre transmitido no campo "RPC *version*". A seguir pode-se ver os campos "*program number*", "*version number*" e "*procedure number*" que unicamente identificam um procedimento específico no servidor. Os campos "*credentials*" e "*verifier*" são usados para fins de autenticação e não serão tratados em detalhes neste trabalho. Na comunicação entre um cliente e o portmapper/rpcbind, que é mais interessante para este trabalho, não existe nenhum esquema de autenticação, no modo de operação normal, sendo usado. Embora estes campos tenham tamanhos variáveis, o número de *bytes* usados é codificado como parte do campo: os primeiros quatro *bytes*, em cada um destes dois campos, representam o tamanho total dos mesmos. Além disso, mesmo que nenhum esquema de autenticação esteja sendo usado, este campo também deve transmitir um valor inteiro 0, que indica a ausência de um esquema de autenticação.

Finalmente, tem-se o campo "*procedure parameters*" com os parâmetros que devem ser passados ao procedimento. O formato deste campo depende da definição de cada procedimento remoto. Deve-se notar aqui que este campo também tem tamanho variado. Para o caso de um datagrama UDP não existem problemas, pois o tamanho deste campo é igual ao tamanho total do pacote UDP subtraído do comprimento total de todos os outros campos mostrados acima. Entretanto, quando TCP estiver sendo usado, um campo extra de quatro *bytes* entre o cabeçalho TCP e o XID é introduzido, codificando o tamanho total da mensagem RPC.

A Figura 2 mostra o formato de uma mensagem *reply* também encapsulada em um pacote UDP. O XID é copiado da mensagem *call* correspondente, como já foi discutido nesta seção. O campo seguinte tem valor 1 indicando que a mensagem se trata de um *reply*. A versão do protocolo RPC é 2, como no caso das mensagens *call*. O campo "*status*" recebe o valor 0 se a mensagem foi aceita (ela pode ser rejeitada, no caso de se estar usando outra versão do RPC ou de alguma falha na autenticação, e o valor atribuído, neste caso, será 1). O "*verifier*" tem o mesmo papel do caso anterior para mensagens *call*. O campo "*accept status*" recebe o valor 0 quando o procedimento é executado com sucesso. Neste caso, um valor diferente de zero identifica um determinado caso de erro, como por exemplo, um número de procedimento inválido sendo enviado pelo cliente. Como em mensagens *call*, se o protocolo de transporte TCP estiver sendo usado, um campo extra de quatro *bytes* é introduzido entre o cabeçalho TCP e o XID. Isto é útil para o cliente descobrir o tamanho correto do campo "*procedure results*", com os resultados retornados pelo procedimento.

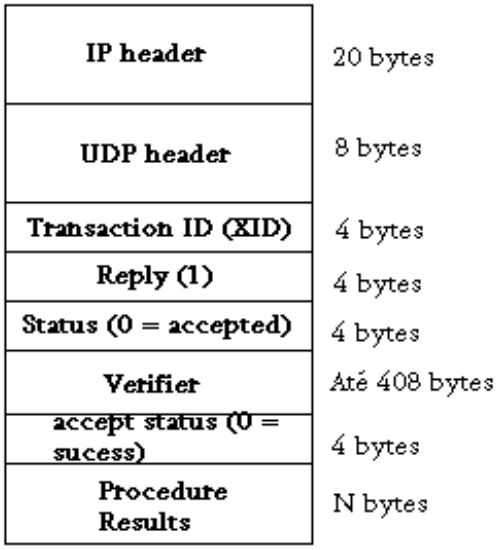

Figura 2: Mensagem reply com UDP

Os campos destas mensagens são codificados usando o padrão de representação XDR, possibilitando que clientes e servidores RPC usem arquiteturas diferentes. XDR define como os vários tipos de dados são representados e transmitidos em uma mensagem RPC (ordem dos *bits*, ordem dos *bytes*, etc.). Portanto, o transmissor cria mensagens codificando todos os dados usando XDR e, do outro lado, o receptor decodifica os dados da mensagem usando este mesmo padrão de representação. Nas figuras 1 e 2, os campos usados transmitem principalmente valores inteiros. Portanto, pode-se concluir que um inteiro em XDR é codificado usando quatro *bytes* de dados.

5.2 Estendendo o módulo de Connection Tracking

Inicialmente foi implementado um módulo auxiliar (ip_conntrack_rpc) para inspecionar o andamento de todas as tentativas de comunicação com o portmapper/rpcbind. Somente mensagens RPC, usando UDP ou TCP, requisitando a execução de procedimentos PMAPPROC_GETPORT e suas respectivas respostas são acompanhadas pelo módulo. O objetivo deste módulo é manter uma lista interna (lista de conexões ou sessões RPC aguardadas) com informações de estados extraídas da parte de dados dos pacotes e permitir que um outro módulo (ipt_rpc_known), estendendo o módulo de filtragem Iptables, possa verificar esta lista e, com base nestas informações, fazer a filtragem corretamente dos serviços baseados em RPC. Cada entrada na lista, representada por uma estrutura de dados (expect_rpc), mantém: os endereços IP do cliente e do servidor RPC, o protocolo a ser usado na comunicação, o número da porta usada pelo servidor RPC e um *timeout* associado à entrada.

Como visto anteriormente, o funcionamento do protocolo RPC inicia quando o cliente se comunica com o portmapper/rpcbind para determinar o número da porta usada por um determinado servidor RPC naquela máquina. Nesta comunicação, o cliente passa como parâmetros o número de programa do serviço, o protocolo e a versão desejada. O módulo Conntrack responsável pelo tratamento do RPC acompanha todas estas requisições. Ele, na verdade, cria uma lista de requisições, para manter o histórico de todas as consultas ao portmapper/rpcbind. Cada entrada na lista de requisições (estrutura de dados request_p) mantém: o endereço IP do cliente, o XID, o número de porta do cliente, o protocolo a ser usado na comunicação com o servidor RPC e um *timeout* associado à entrada. Este timeout evita que consultas forjadas criem um número indefinido de novas entradas em ataques de DoS (*Denial of Service*), consumindo recursos na máquina filtradora.

Caso este servidor esteja registrado, o portmapper/rpcbind retornará o número da porta usada pelo servidor naquela máquina. Quando esta mensagem *reply* chega ao módulo Conntrack, este pode verificar se existe uma entrada na lista de requisições e, caso exista, pode criar uma entrada na lista de conexões ou sessões RPC aguardadas. Nesta verificação, além das informações mantidas na lista de requisições, o módulo observa o XID, o endereço IP e a porta do cliente nestas mensagens. Isso é necessário, pois a informação do protocolo a ser usado não está disponível na mensagem *reply*. Para que uma entrada seja criada na lista de conexões ou sessões RPC aguardadas esta informação também é necessária.

Se não existir uma entrada na lista de requisições correspondendo à mensagem *reply*, o módulo Conntrack simplesmente descarta a mensagem para um acompanhamento mais correto das comunicações. O leitor pode imaginar uma situação em que o portmapper/rpcbind, por algum motivo, atrasa a execução do procedimento PMAPPROC_GETPORT e o cliente retransmite a requisição. Pode-se supor então um cenário onde o timeout para entrada na lista de requisições expira e a mensagem de requisição retransmitida pelo cliente é perdida em trânsito e, portanto, não seria vista pelo módulo Conntrack. Caso esta mensagem *reply* chegue ao cliente, ele tentará fazer a comunicação com o servidor RPC normalmente. Entretanto, esta comunicação não será permitida devido a inexistência de uma entrada para ela na lista de conexões ou sessões RPC aguardadas.

Um outro aspecto importante é o tratamento das possíveis comunicações subseqüentes dos clientes com os servidores RPC. O *timeout* associado a cada entrada na lista de conexões ou sessões aguardadas tem como objetivo básico permitir futuras tentativas de comunicação de clientes usando sua respectiva *cache* de respostas

anteriores do portmapper/rpcbind. Se a entrada fosse excluída imediatamente após o fim da comunicação, futuras tentativas não seriam permitidas. Obviamente, clientes bem implementados devem prover um esquema de recuperação para falhas deste tipo. Esta falha seria similar ao caso de um servidor registrado ser reinicializado e o cliente, usando as informações da cache, tentar usar o serviço.

5.3 Estendendo o Iptables

Como já foi mencionado, o módulo de filtragem, nas séries 2.3 e 2.4 do *kernel*, baseia sua decisão também em informações de estados dos pacotes dentro de suas respectivas sessões ou conexões. Um módulo estendendo o Iptables foi criado para tratar todos os pacotes RPC com estado NEW (iniciando uma sessão ou conexão) chegando ao módulo de filtragem. Desta forma, o Iptables permite todas as tentativas de comunicação com o portmapper/rpcbind com consultas aos números de portas sendo usadas pelos servidores registrados. Portanto, não há a necessidade de uma regra permitindo explicitamente a comunicação com o servidor de nomes do RPC. Este novo módulo do Iptables também é o responsável em verificar as entradas da lista de conexões e sessões aguardadas e permitir que clientes se comuniquem normalmente com servidores RPC.

5.4 Testes de Sanidade e Cuidados Adicionais com Fragmentos e Pacotes Truncados

Para um correto acompanhamento das mensagens RPC, alguns cuidados devem ser tomados com respeito à sanidade dos pacotes. Além disso, os módulos devem tomar cuidados extras para evitar que pessoas mal intencionadas possam subverter o mecanismo de filtragem de pacotes. O módulo Conntrack, antes de inspecionar a parte de dados de todos os pacotes RPC, verifica se há uma má formação e calcula o *checksum* do cabeçalho de transporte no pacote. Se UDP estiver sendo usado, esta última operação somente é feita se o campo de *checksum* tiver um valor diferente de zero, indicando que ele foi calculado, já que a checagem em UDP é opcional e percebida através do valor diferente de zero.

Pacotes e fragmentos de pacotes cuidadosamente construídos foram previstos durante a implementação, tendo-se em mente, principalmente, a vulnerabilidade recente encontrada no FW-1 e Cisco-PIX no suporte ao FTP. Com este problema, um atacante é capaz de subverter o filtro de pacotes com estados. Para isto, bastava forçar que os pacotes do protocolo FTP fossem devidamente truncados e, explorando um problema existente na inspeção da parte de dados, abrir brechas no *firewall*.

Como já mencionado, as mensagens RPC trocadas entre clientes e o portmapper/rpcbind são extremamente pequenas. Desta forma, cabem facilmente em qualquer MTU (*Maximum Transmission Unit*) existente nas interfaces de rede atuais. Desta forma, pacotes RPC fragmentados são tratados como um comportamento anormal e simplesmente ignorados pelo módulo Conntrack. Além disso, pacotes RPC, onde a parte de dados é maior que o esperado, são imediatamente descartados.

Se um cliente usar TCP para a comunicação com o portmapper/rpcbind, os valores dos atributos da conexão devem ser sempre suficientemente grandes para não truncar a parte de dados nos pacotes. Em geral, o valor mínimo atribuído ao parâmetro MSS (*Maximum Segment Size*), por exemplo, é idêntico à MTU usada pelo meio físico para evitar a indesejável fragmentação de pacotes. Pode-se concluir, com isso, que não há boas razões para usar valores muito pequenos para este parâmetro e, portanto, qualquer tentativa disto é vista como um ataque contra o filtro de pacotes com estados.

6 – CONCLUSÃO

Este trabalho mostrou que filtros de pacotes com estados oferecem condições para a filtragem de serviços baseados em RPC. Vale notar ainda que soluções semelhantes podem ser adotadas para outros protocolos e serviços mais complexos. Um bom exemplo é o serviço Real Audio, para transmissão de áudio na Internet. Neste serviço, o cliente estabelece uma conexão TCP usando a porta 7070 do servidor. Usando esta conexão, ele envia suas requisições e números de portas UDP que ficarão esperando os dados do servidor. Múltiplas sessões UDP serão usadas para aumentar a vazão, garantindo a qualidade do som. Esta quantidade de portas usadas pode sobrecarregar muito mais rapidamente *proxies* dedicados. Além disso, o aumento no *overhead* pode prejudicar a qualidade do som. Um SPF, por outro lado, pode acompanhar a conexão TCP e permitir as sessões UDP muito facilmente. Vale ressaltar, porém, que agentes *proxy* dedicados devem ser adicionados sempre que o fluxo de dados para algum serviço precisar ser remontado/normalizado e inspecionado mais detalhadamente.

AGRADECIMENTOS

Os autores agradecem ao CNPq, à FAEP-UNICAMP pelo financiamento ao trabalho e, principalmente, ao Rusty Russel por todos os esclarecimentos e sugestões a respeito da nova

infra-estrutura de tratamento e filtragem de pacotes no *kernel* do Linux.

REFERÊNCIAS BIBLIOGRÁFICAS

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

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

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

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

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

[6] GONÇALVES, Marcus. *Firewalls Complete*. McGraw-Hill, 1997.

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

[8] 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.

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

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

[11] LIMA, Marcelo Barbosa; GEUS, Paulo Lício de. *Comparação entre Filtro de Pacotes com Estados e Tecnologias Tradicionais de Fiewall*. SSI99, São José dos Campos, 1999.

[12] LIMA, Marcelo Barbosa. *Provisão de Serviços Inseguros Usando Filtros de Pacotes com Estados*. Campinas: IC-Unicamp, Setembro 2000.

[13] Netfilter *Home Page.* URL: http://netfilter.kernelnotes.org. Julho, 2000.

[14] STEVENS, W. Richard. *TCP/IP Illustrated.* Addison Wesley, 1994.

[15] Sun Microsystems. RPC: Remote Procedure Call Protocol Specification. Version 2. RFC 1057, Junho, 1988.