NFS Guard: Uma Solução de Segurança para o Protocolo NFS

Guilherme Cesar Soares Ruppert

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

NFS Guard: Uma Solução de Segurança para o Protocolo NFS

Guilherme Cesar Soares Ruppert[1]

Fevereiro de 2006

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus (Orientador)
- Prof. Dr. Fabrício Sérgio de Paula
  Ciência da Computação, Universidade Estadual do Mato Grosso do Sul
- Prof. Dr. Ricardo Dahab
  Instituto de Computação, Universidade Estadual de Campinas
- Prof. Dr. Rodolfo Jardim de Azevedo (Suplente)
  Instituto de Computação, Universidade Estadual de Campinas

[1]**Financiado por FAPESP - Fundação de Amparo à Pesquisa do Estado de SP**

Substitua pela ficha catalográfica

NFS Guard: Uma Solução de Segurança para o Protocolo NFS

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Guilherme Cesar Soares Ruppert e aprovada pela Banca Examinadora.

Campinas, 10 de Fevereiro de 2006.

Prof. Dr. Paulo Lício de Geus (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Substitua pela folha com a assinatura da banca

À Vera, Ernesto e Heloisa.

"O único homem que está isento de erros
é aquele que não arrisca acertar."
- *Albert Eistein*

Agradecimentos

A Deus, que se fez presente em todos os momentos de minha vida.

Aos meus pais, Vera e Ernesto, por todo amor infinito e incondicional, por estarem sempre ao meu lado e, sobretudo, por serem meus maiores exemplos de vida.

À minha namorada, Heloisa, pelo amor sincero e por todo o apoio e incentivo nos momentos difíceis.

À minhas irmãs, Gláucia e Lídia, pela união e pelos incontáveis momentos de alegria.

Às minhas avós Lourdes, Maria Joanna pelo carinho, meus tios Vera e Paulo e a todos os meus familiares, que sempre contribuiram de alguma forma motivando o meu trabalho.

Ao meu orientador, Paulo Lício de Geus, pelo apoio não só profissional como pessoal, e também pelas inúmeras oportunidades proporcionadas.

Aos integrantes do LAS, Cleymone, João Porto, Fernando, Felipe, Diogo, Arthur, Eduardo, Martim e Helen, aos antigos e novos membros por nossas conversas de laboratórios, seus conselhos e suas contribuições, e pela amizade estabelecida.

À todos os colegas, professores e funcionários do Instituto de Computação.

À Fundação de Apoio à Pesquisa do Estado de São Paulo (FAPESP) pelo apoio financeiro através da bolsa concedida.

Resumo

Os sistemas de arquivos distribuídos são um componente importante em praticamente todas as redes coorporativas, acadêmicas e até mesmo domésticas. Neste cenário, o NFS (*Network File System*) é um dos sistemas de maior importância, não só pela importância histórica em ter sido um dos pioneiros, mas por ser largamente utilizado em todo o mundo.

Porém, o NFS foi projetado com recursos insuficientes de segurança. A versão 3 do NFS, que é a versão utilizada atualmente em ambientes de produção, apresenta um nível de segurança praticamente nulo. Mais recentemente, foi elaborada a especificação da versão 4 do protocolo que corrige adequadamente os problemas de segurança. Contudo, o protocolo apresenta um grau de complexidade extremamente elevado e, devido a isso, poucos estão dispostos a enfrentar sua implementação. Atualmente, existem apenas algumas implementações preliminares deste protocolo e as mesmas não se encontram completas e estáveis. Logo, a migração para o NFSv4 a curto prazo é inviável.

Neste trabalho foi realizado um amplo estudo dos diversos aspectos envolvidos com a segurança de sistemas de arquivos distribuídos, em especial o NFS. Foram levantadas as deficiências do NFS e foram analisados diversos trabalhos envolvendo segurança no compartilhamento de arquivos.

Este trabalho resultou no projeto e implementação do **NFS Guard**, uma ferramenta para agregar segurança ao NFS versão 3 de forma simples e totalmente compatível com as redes NFS já instaladas. O NFS Guard utiliza clientes e servidores NFS padrão e não requer recompilação de nenhum software já instalado, podendo ser utilizado inclusive em sistemas de código fechado. Para isso foi desenvolvida uma técnica de Kerberização utilizando interceptação de pacotes, que também pode ser utilizada para agregar segurança à outros serviços além do NFS.

Abstract

Distributed File Systems are an important component in practically every corporate, academic or even home network. In this scenario, NFS (Network File System) is one of the most important of these systems, not only due to its historical importance for being one of the pioneers, but also for being largely used all over the world.

However, NFS was designed with insufficient security features. NFS version 3, which is the version currently being used in production environments, presents an almost null security level. More recently, the version 4 of the protocol was designed,among other thigs, to tackle the security issues. Nevertheless, this protocol presents an extremely high level of complexity, and consequently, a few want to face its implementation. At the moment, a few preliminary implementations are available, but usually incomplete and unstable. Hence, the migration to NFSv4 in a short term is not feasible.

In this work we accomplished a wide study of many aspects involved with distributed file system security, specially with regards to NFS. NFS's shortcoming were identified e many works involving security in file sharing were analyzed.

This work resulted in the design and implementation of **NFS Guard**, a tool to add security do NFS version 3 in a simple way and totally compatible with installed NFS networks. NFS Guard uses standard NFS client and server and does not require recompilation of any installed software, so it can be used with closed source systems. To accomplish that, a Kerberization technique was developed using packet interception, which can also be used to add security to other services besides NFS.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Um sistema de arquivos distribuído (ou sistema de arquivos em rede) é uma aplicação cliente/servidor que permite que os clientes acessem arquivos armazenados no servidor de forma transparente como se fossem arquivos locais em seus computadores. O objetivo destes sistemas é prover compartilhamento de arquivos com transparência de localização. Existem vários sistemas que provêem compartilhamento de arquivos, tais como FTP, HTTP, SCP, redes *Peer-to-Peer*, bancos de dados, dentre outros. Porém, nestes sistemas não há transparência de localização, pois tanto o usuário como a aplicação têm ciência de que estão lidando com arquivos remotos de um servidor específico. Já em um sistema de arquivo distribuído, qualquer aplicação que trabalhe com arquivos locais irá funcionar com arquivos remotos sem qualquer modificação. Neste caso, a transparência de localização provida faz com que a aplicação sequer tenha ciência da localização real deste arquivo, utilizando-o como um simples arquivo local.

Os sistemas de arquivos distribuídos são um componente importante em praticamente todas as redes coorporativas, acadêmicas e até mesmo domésticas. Os indiscutíveis benefícios destes sistemas os tornaram tão utilizados que é quase impossível encontrar uma organização que não utilize esse mecanismo para compartilhar arquivos entre os usuários.

Neste cenário, o NFS (*Network File System*) se destaca como um dos sistemas de maior importância, não só pela importância histórica em ter sido um dos pioneiros, mas por ser largamente utilizado em todo o mundo. Nos sistemas operacionais Unix em especial, o NFS é onipresente sendo considerado o sistema padrão para compartilhamento de arquivos em redes Unix.

Porém, o NFS nasceu em uma época que segurança computacional não era um requisito importante, portanto o protocolo foi projetado com recursos insuficientes de segurança como iremos demonstrar neste trabalho. A versão 3 do NFS, que é a versão utilizada atualmente em ambientes de produção, apresenta um nível de segurança praticamente nulo. Mais recentemente, foi elaborada a especificação da versão 4 do protocolo, que corrige adequadamente os problemas de segurança. Contudo, a nova versão apresenta um grau de complexidade extremamente elevado e, devido a isso, poucos estão dispostos a enfrentar sua implementação. Atualmente, existem apenas algumas implementações preliminares deste protocolo e as mesmas não se encontram completas e estáveis. Logo, a migração para o NFSv4 a curto prazo é inviável.

1.1 Motivação

A motivação deste trabalho partiu da experiência prévia com administração de redes Unix que utilizam o NFS. Os problemas de segurança do NFS foram vivenciados na prática e também constavam na literatura de administração e seguraça ([Gar96] e [Ste01]). Estudos preliminares mostraram que não havia uma solução estabelecida para estes problemas e que, portanto, havia espaço para um trabalho de mestrado nesse tópico.

Neste trabalho foi realizado um amplo estudo dos diversos aspectos envolvidos com a segurança de sistemas de arquivos distribuídos, em especial o NFS. Foram levantadas as deficiências do NFS e foram analisados diversos trabalhos envolvendo segurança no compartilhamento de arquivos.

Este trabalho resultou no projeto e implementação do **NFS Guard**, uma ferramenta para agregar segurança ao NFS versão 3 de forma simples e totalmente compatível com as redes NFS já instaladas. O NFS Guard utiliza clientes e servidores NFS padrão e não requer recompilação de nenhum software já instalado, podendo ser utilizado inclusive em sistemas de código fechado. Para isso foi desenvolvida uma técnica de *Kerberização* (agregar suporte ao Kerberos) utilizando interceptação de pacotes, que também pode ser utilizada para agregar segurança à outros serviços além do NFS.

1.2 Organização do trabalho

Baseado nas etapas do desenvolvimento deste trabalho, os diversos aspectos envolvidos na elaboração de uma solução foram agrupados na seqüência apresentada a seguir.

Inicialmente é apresentado, no capítulo 2, o protocolo NFS sob suas diversas versões. É feita uma análise focada nos aspectos de segurança do NFS, em especial apontando as falhas de segurança que desejamos contornar.

No Capítulo 3 são apresentados diversos sistemas de arquivos distribuídos de possuem recursos de segurança, alguns deles baseados no NFS, outros não. Vários desses trabalhos correlatos contribuíram com idéias positivas ou negativas e serviram de inspiração para o desenvolvimento da solução.

Já no Capítulo 4, apresentamos a solução proposta para o problema. São descritos os requisitos estabelecidos para o projeto. A arquitetura projetada é apresentada e as decisões de projeto são justificadas. Neste Capítulo abordamos apenas a especificação da solução proposta, deixando de lado os detalhes de implementação.

O Capítulo 5 apresenta a implementação da solução proposta. São descritas as técnicas, dificuldades e decisões tomadas para a implementação no Linux da solução proposta no Capítulo 4.

Por fim, são apresentadas no Capítulo 6 algumas considerações e conclusões gerais sobre todas as etapas envolvidas no desenvolvimento deste trabalho, além de algumas sugestões para trabalhos futuros.

Capítulo 2

Network File System (NFS)

O NFS - Network File System (Sistema de Arquivos em Rede) foi originalmente projetado e implementado pela Sun Microsystems e endossado por companhias como: IBM, Hewlett-Packard, Apollo Computer, Apple Computer e muitos outros fornecedores de sistemas Unix.

Trata-se de um protocolo para acesso a arquivos remotos. Esse protocolo é certamente o mais utilizado para se obter compartilhamento de arquivos através da rede em ambientes Unix, sendo que todos os sistemas Unix possuem implementação desse protocolo.

A idéia principal do projeto do NFS é prover acesso transparente para o usuário aos arquivos remotos, ou seja, as aplicações acessam arquivos que estão remotos utilizando a mesma semântica Unix de acesso a arquivos locais. Para alcançar esse objetivo, o sistema de arquivos remoto é montado em algum ponto da hierarquia local de arquivos e, a partir daí, é tratado como um arquivo local.

Existem atualmente quatro versões do protocolo NFS. A primeira versão do NFS foi um protótipo que existiu apenas internamente à Sun Microsystems e nunca foi disponibilizada publicamente. A segunda versão (NFSv2) [SM89] desse protocolo foi implementada e disponibilizada inicialmente no SunOS 3.2 em 1985. O NFSv2 foi amplamente utilizado, no entanto, essa versão apresentava algumas limitações, que só poderiam ser contornadas com a especificação de uma nova versão. Problemas como: limitação do tamanho do arquivo para 4GB, performance inadequada devido à escrita síncrona e falta de garantia de consistência levaram à necessidade de uma nova versão.

Em Junho de 1995 foi concluída a especificação do NFSv3 [Cal95] e, em alguns anos,

todas as implementações de Unix suportariam esta nova versão do protocolo. Porém grande parte das implementações atuais do NFSv3 não implementa toda a RFC. Um exemplo disso é a implementação para o Linux, cujo suporte de NFS sobre TCP ainda encontra-se em fase experimental. Além disso, a especificação da versão 3 do NFS já contemplava alguns mecanismos de segurança como autenticação via Kerberos e Diffie-Hellman, porém sem caráter mandatório. O resultado disso é que a grande maioria das implementação não colocou esses quesitos de segurança em prática.

O NFSv3 é a versão existente nos sistemas em produção atualmente, incluindo Linux, FreeBSD e Solaris.

Devido à necessidade de uma melhor performance através da Internet, e também de segurança baseada em esquemas de criptografia forte, um comitê de pesquisadores e empresas começou a trabalhar na especificação da versão 4 do NFS. A especificação final foi concluída em dezembro de 2000 [She00], mas passou por uma revisão [She03] em abril de 2003, apresentando mudanças radicais no protocolo da versão anterior, principalmente no aspecto da segurança. Algumas RFCs correlatas ainda estão em desenvolvimento para padronizar assuntos como padronização de ACLs dentro do NFSv4, mecanismo de cache de credeciais, migração de hierarquias entre servidores etc.

Existem atualmente apenas três projetos de implementações deste protocolo, sendo que um deles é considerado a implementação de referência. Esta implementação foi incluída no kernel oficial do Linux a partir da versão 2.6, porém ainda está incompleta e instável.

2.1 NFSv2 E NFSv3

Tanto a versão 3 como a versão 2 do NFS são totalmente baseadas nos protocolos RPC (Remote Procedure Call) e XDR (eXternal Data Representation) da Sun Microsystems. Essas duas versões do NFS são bastante similares porque, na terceira versão, poucas modificações foram feitas por receio de alterar radicalmente um protocolo já consagrado que vinha funcionando satisfatoriamente.

O NFS é na verdade dividido em dois protocolos: o Mount, e o NFS propriamente dito. O primeiro é responsável pela negociação inicial entre cliente e servidor. Através do mount, o cliente pode determinar quais sistemas de arquivos estão disponíveis para montagem e obter um descritor de arquivos, que é utilizado para referenciar esses arquivos

remotamente. O segundo protocolo permite que o cliente liste o conteúdo dos sistemas de arquivos exportados e obtenha recursivamente os descritores dos outros arquivos, podendo assim criar, modificar e apagar arquivos.

O NFS baseia-se no modelo de segurança do RPC para garantir a segurança de seus serviços. O servidor NFS checa as permissões utilizando as credenciais da informação de segurança do RPC contida em cada requisição remota.

Iremos agora analisar essas duas versões do NFS levando em conta cada um dos seguintes tópicos de segurança: integridade, privacidade, não-repúdio, controle de acesso e autenticação.

2.1.1 Integridade, Privacidade e Não-repúdio

Os protocolos NFSv2 e NFSv3, da forma em que foram especificados, utilizam o protocolo RPC versão 2, que não possui qualquer mecanismo que possibilite a garantia da integridade, privacidade ou não-repúdio dos dados. Existem dois campos (credential e verifier) no cabeçalho do RPC que são dedicados à autenticação, porém todos os dados trafegam pela rede de forma não cifrada.

Portanto, nas versões 2 e 3 do NFS, um atacante pode realizar um ataque conhecido como sniffing, que consistem simplesmente em capturar todo o tráfego no cabo de rede e, dessa forma, conseguirá ter acesso de leitura a todos os dados (arquivos e diretórios) acessados naquele instante. De posse das requisições e respostas capturadas em texto claro, o atacante terá o trabalho de interpretar toda a comunicação e remontar os arquivos transferidos. Porém, esse é um ataque bastante viável principalmente para arquivos texto, que são arquivos de maior inteligibilidade.

O atacante também poderá realizar um ataque à integridade da informação. Utilizando um conhecido ataque chamado man-in-the-middle, o adulterador poderia interceptar uma mensagem RPC e dessa forma alterar o conteúdo das requisições e respostas do NFS. Com esse ataque pode-se, por exemplo, alterar o conteúdo das chamadas WRITE() em andamento naquele instante e, desse modo, alterar o conteúdo de arquivos gravados no servidor. Algoritmos de detecção de erro como CRC-32, por exemplo, não são úteis nesse caso, pois o adulterador pode refazer o checksum com o pacote modificado. Seria necessário fazer uso de algum protocolo criptográfico para autenticar uma assinatura da mensagem. Dessa forma, ter-se-ia certeza de que aquela assinatura foi gerada pelo

remetente e, portanto, a integridade da mensagem poderia ser garantida.

O outro problema de segurança apontado é a falta de garantia do não-repúdio. Nas versões 2 e 3 do NFS, um usuário pode negar que tenha realizado certas operações no servidor. Uma situação emblemática desse problema é o caso de um usuário que deleta arquivos importantes do servidor e que poderá negar que o tenha feito. Isso é causado por diversos motivos: falta de registros de log no servidor; mecanismos fracos de autenticação, como será visto mais adiante, e possibilidade de ataques à integridade dos dados.

2.1.2 Controle de Acesso

O NFSv2 possui um mecanismo simples de controle de acesso baseado nos bits de permissões (atributos) do arquivo. Verificando os bits do arquivo que está sendo acessado, o cliente consegue saber se poderá ou não efetuar a operação. Este método permite um nível bastante razoável de controle de acesso, porém apresenta algumas limitações indesejáveis. Este método não suporta o controle de permissões através de ACLs e dificulta a interoperabilidade com sistemas não-Unix que possuem atributos de permissões diferentes.

No NFS versão 2 a única forma confiável de se checar se uma operação é realmente permitida é tentar realizá-la e checar se a operação tem sucesso. A versão 3 do NFS introduz o procedimento ACCESS() que permite que o cliente possa perguntar ao servidor se uma dada operação é permitida, antes de executá-la. Isso é útil em sistemas operacionais, como o Unix, onde as permissões são checadas quando um arquivo é aberto. Com esse procedimento o servidor pode utilizar qualquer política de permissões (inclusive ACL´s) pois agora a tarefa de checar as permissões são de responsabilidade do servidor.

2.1.3 Autenticação

Conforme já foi dito, o NFS utiliza o protocolo RPC e existem dois campos (credential e verifier) no cabeçalho do RPC que são dedicados à autenticação. É através desses campos que o NFS provê mecanismos de autenticação tanto para o cliente quanto para o servidor.

Diversos protocolos de autenticação podem ser suportados pelo RPC, e um campo no cabeçalho RPC chamado esquema de autenticação (authentication flavour) indica qual protocolo será utilizado. A tabela 2.1 mostra os protocolos existentes.

#	Nome	Descrição
1	AUTH_NONE	Ausência de autenticação. Acesso anônimo.
2	AUTH_SYS	O servidor simplesmente confia no UID e GID fornecidos pelo cliente.
3	AUTH_DH	Protocolo baseado no DES e Diffie-Hellman para troca de chaves.
4	AUTH_KERB	Baseado nos protocolos Kerberos 4 e DES para troca de chaves.
6	RPCSEC_GSS	Baseado na GSS_API

Tabela 2.1: Esquemas de Autenticação do NFS

A única implementação que suporta todos esses esquemas é o software da Sun. Todas as outras implementações estáveis (inclusive a do Linux) implementam somente AUTH_NONE e AUTH_SYS.

Os esquemas AUTH_DH e AUTH_KERB [Mic88] também são conhecidos por seus nomes comerciais, respectivamente, *Secure NFS* e o *Kerberized NFS*. Esses esquemas apresentam características de segurança bastante interessantes.

O AUTH_DH utiliza o algoritmo de Diffie-Hellman para estabelecer uma chave de conversação DES. Por esse motivo, este esquema também é conhecido por AUTH_DES. Esse mecanismo é implementado na camada RPC (Secure RPC) que forma a base para o NIS+. Ao contrário do AUTH_SYS, o AUTH_DH utiliza o campo de verificador do cabeçalho RPC, e através dele o servidor pode validar a credencial do cliente (e vice-versa). O AUTH_DH possui diversas deficiências de segurança. Primeiramente, ele não especifica um método seguro para obter a chave pública do servidor. As primeiras implementações utilizavam mapas do NIS para distribuir chaves, porém, como esse serviço não possui segurança, um atacante poderia facilmente impersonar o servidor NIS e distribuir chaves falsas. Isso seria resolvido com a chegada do NIS+, que é um serviço de nomes seguro que também usa o AUTH_DH. Porém, foi demonstrado [LaMacchia91] que a implementação da autenticação DH era fraca devido a escolha de uma chave pequena (192 bits). Até mesmo para os computadores da época, seria possível derivar uma chave privada a partir de uma chave pública em questão de minutos. Outra deficiência do AUTH_DH é a utilização do algoritmo DES, que é um algoritmo considerado inseguro para o estado da arte atual da computação. Em 1999, um computador distribuído quebrou uma chave DES em

aproximadamente 22 horas no concurso *DES Challenge III*.

O AUTH_KERB é um esquema do Sun RPC que realiza autenticação utilizando o sistema Kerberos versão 4 desenvolvido pelo MIT. Trata-se de um esquema de autenticação similar ao AUTH_DH, usando o algortimo DES para cifragem, porém o AUTH_KERB utiliza tickets ao invés de chaves públicas. Este esquema também possui deficiências de segurança devido às falhas conhecidas do Kerberos 4 [SB91].

Além dos problemas de segurança citados, tanto o AUTH_DH como o AUTH_KERB sofreram resistência pois havia uma grande dificuldade de implementação para os administradores de sistemas e também porque a Sun Microsystems, que era a empresa que desenvolvia o Secure NFS e o Kerberized NFS, demorava em dar suporte desses protocolos aos novos sistemas operacionais que lançava. Esses fatores, somados a problemas com exportação de criptografia americana e ao grande impacto da criptografia na performance para os equipamentos da época, fizeram com que esses esquemas caíssem em desuso. Como a RFC do NFS não especificava esses esquemas mais seguros com caráter mandatório, quase nenhum outro sistema operacional (exceto o *SunOS*) implementou os mesmos.

O último esquema da tabela é o RPCSEC_GSS, [Eis97] que é um esquema que surgiu posteriormente aos demais. Este esquema é utilizado pelo NFSv4 e será visto mais adiante.

Portanto, atualmente o único esquema de autenticação utilizado e presente em todas as implementações de NFS é o AUTH_SYS (também conhecido como AUTH_UNIX). Iremos agora analisar as falhas desse esquema para autenticar cada um dos personagens ligados ao serviço NFS: o cliente, o servidor e o usuário.

Autenticação do Cliente

O esquema AUTH_SYS provê um mecanismo extremamente fraco para autenticar a máquina cliente. Neste esquema, o servidor autentica o cliente apenas comparando o endereço IP da máquina que fez a requisição com uma lista de máquinas permitidas.

Vamos primeiramente dar uma olhada no funcionamento do NFS, mais especificamente no protocolo Mount. Suponha que o cliente deseja, por exemplo, montar o diretório /home da máquina aracaju (servidor) em /users na máquina blumenau (cliente), o usuário poderá executar o seguinte comando na máquina blumenau:

```
# mount -t nfs aracaju:/home /users
```

O programa mount irá conectar-se com o programa servidor remoto chamado mountd na máquina aracaju via RPC. O servidor irá verificar em /etc/exports se o IP da máquina blumenau tem permissão para montar o diretório em questão, e caso tenha, retorna a ela um descritor de arquivos. Este descritor de arquivos será usado em todas as requisições posteriores aos arquivos sob o diretório users.

Nesta etapa do protocolo existe um problema de segurança referente à autenticação da máquina cliente. A autenticação é baseada apenas no endereço IP do cliente mas sabemos que uma máquina pode facilmente se passar por outra bastando reconfigurar o endereço IP (com o endereços de alguma máquina desligada) ou então realizando um ataque de spoofing (falsificação do endereço IP). Outras técnicas de ataque (como negação de serviço, alteração de rotas etc) podem ser necessárias para que o IP spoofing tenha sucesso. Isso é bastante inseguro, por exemplo, em um cenário onde um atacante pode conectar seu laptop à rede. Basta ele configurar seu endereço IP para o endereço de uma máquina desligada para poder montar qualquer filesystem que a máquina impersonada teria acesso.

Autenticação do Servidor

O NFSv3 utilizado atualmente não provê nenhum mecanismo que permita que o cliente autentique o servidor. O cliente não tem como descobrir se o servidor é, de fato, o servidor legítimo, ou se alguma outra máquina está se passando pelo servidor.

Existem diversas formas que o atacante poderia utilizar para impersonar o servidor e explorar a falta de autenticação do NFS. Técnicas como IP spoofing, DNS spoofing e man-in-the-middle, são comprovadamente eficientes para tal objetivo. O atacante poderia utilizar um servidor falso, por exemplo, para receber arquivos ou para disponibilizar arquivos falsos.

Autenticação do Usuário

Quando alguém acessa um arquivo remoto através do NFS, o kernel envia uma chamada RPC para o daemon nfsd na máquina servidora. Esta chamada leva como parâmetros: o descritor de arquivos, o identificador de usuário (UID) e de grupo (GID). Eles são usados

para controlar os direitos de acesso sobre um determinado arquivo. Nessa etapa existe outro problema de segurança: a autenticação do usuário. O servidor não se certifica se o UID recebido na requisição é realmente o UID do usuário que está requisitando a operação. O servidor simplesmente confia na informação contida na requisição e executa a operação solicitada. Existem dois cenários bastante comuns onde essa falta de autenticação representa muita dor de cabeça para os administradores.

O primeiro cenário é uma rede onde os usuários podem ter máquinas pessoais com privilégios de administrador. O servidor NFS possui os homedirs dos usuário A e B. O usuário A na máquina A cria localmente um usuário B e utiliza o comando su para trocar para o usuário B. Nesse momento toda requisição ao NFS será identificada como usuário B e, dessa forma, o usuário A terá acesso a todos os arquivos do usuário B.

O segundo cenário é mais alarmante ainda. Em qualquer rede que utilize o serviço NFS, qualquer usuário pode ter acesso total aos arquivos de qualquer outro usuário. Note que neste cenário não há necessidade de ser root (super-usuário). Para realizar isso basta que o usuário A escreva um programa em nível de usuário que envie as mesmas chamadas RPC que o cliente NFS envia, só que se identificando com o UID do usuário B. Como o servidor confia cegamente nas mensagens que recebe, ele aceitará a mensagem e executará a requisição com as credenciais falsas. Dessa forma, o usuário A terá acesso a todos os arquivos do usuário B.

2.1.4 Considerações finais sobre NFSv2 e NFSv3

Existe ainda um outro problema de segurança, que não se encaixa em nenhum dos quesitos anteriores e diz respeito à integração do NFS com firewalls. O servidor NFS não utiliza um número de porta fixo, ou seja, não se pode saber de antemão em qual porta o servidor aguarda requisições. Apesar desse serviço estar sempre executando na porta 2049, é possível que ele esteja aguardando em outra porta. Logo, é necessário consultar um outro serviço chamado portmapper para se descobrir em qual porta está registrado o serviço NFS. O grande problema disso é que, como o número da porta é variável, fica extremamente difícil projetar regras de filtragem do tráfego de NFS. Sistemas auxiliares seriam necessários para se permitir tráfego NFS através de filtros, como o módulo desenvolvido por Marcelo Lima para incorporação ao framework netfilter/iptables do Linux [Lim00].

Como pudemos notar pela análise dos quesitos, a situação atual do NFSv3 é extremamente preocupante, pois a segurança desse protocolo é praticamente inexistente. O serviço depende de uma série de outros protocolos (NFS, Mount, Portmap, RPC etc), o que facilita o aparecimento de vulnerabilidades, além de não apresentar mecanismos de segurança adequados e mandatórios.

2.2 NFSv4

Apesar do NFSv3 ter apresentado algumas melhorias em relação à versão 2, o protocolo continuou voltado para redes locais Unix. Em 1993, quando o NFSv3 foi desenvolvido, a Internet estava muito longe se ser o que ela é hoje. Não havia World Wide Web e nem comercio eletrônico. Muita coisa mudou de lá para cá e o NFSv3 não acompanhou esse novo cenário. Algumas modificações para tornar o protocolo mais compatível com a Internet foram introduzidas pelo protocolo WebNFS, que foi uma tentativa de utilizar o NFS para transferir arquivos na Web, através do próprio navegador.

Em 1998 foi formado um grupo de trabalho IETF para trabalhar na versão 4 do NFS e a especificação desse protocolo foi reconhecida como padrão em dezembro de 2000 [She00], mas passou por uma revisão [She03] em abril de 2003.

O NFSv4 apresenta uma ruptura com as versões anteriores. O protocolo foi extremamente modificado, inclusive em questões tidas como alicerces da antiga versão. Exemplo disso é o fato do novo protocolo manter estados das operações no servidor. Na versão antiga, o servidor não mantinha estados (stateless). As operações eram independentes e por isso não se fazia necessário qualquer mecanismo de recuperação de falhas.

Dentre as principais mudanças no protocolo destacam-se:

- O uso do TCP é mandatório, embora continue suportando o UDP.
- Permite chamadas de RPC compostas, ou seja, várias operações em uma mesma mensagem.
- O NFSv4 combina diferentes protocolos (Mount, NLM, etc) em único protocolo.
- Introduz delegação de arquivos, o que permite mecanismos eficientes de cache.

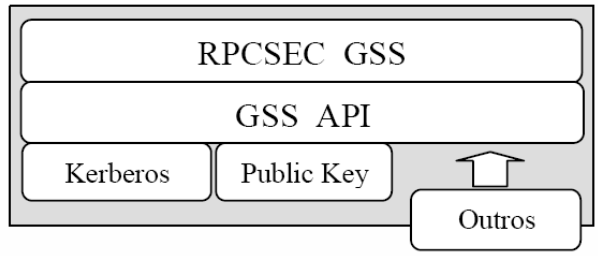

Figura 2.1: O Esquema RPCSEC_GSS

- Diversos mecanismos de segurança usando criptografia forte em caráter mandatório, com suporte à negociação do nível de segurança.

Neste trabalho estaremos analisando apenas as novidades do novo protocolo referentes à segurança.

2.2.1 RPCSEC_GSS

O NFSv4, assim como as outras versões, baseia-se no modelo de segurança do RPC para garantir a segurança de seus serviços, ou seja, utiliza os esquemas de autenticação da tabela 2.1. Porém, o NFSv4 obriga a implementação do esquema RPCSEC_GSS que é baseado na Generic Security Services API (GSS-API) [Lin00].

A GSS-API é uma interface de programação genérica, que tem por objetivo oferecer serviços para construção de aplicações e serviços de segurança. Um dos objetivos desta interface é obter isolamento entre a camada de aplicação e os serviços de segurança propriamente ditos. Desta forma, é possível trocar os algoritmos de criptografia e autenticação de uma aplicação sem alterar seu código. A figura 2.1 mostra que a GSS-API torna o esquema de segurança transparente para a aplicação, no caso, o RPCSEC_GSS.

Diferentemente dos outros esquemas (flavours) que só provêm autenticação, o RPCSEC_GSS pode garantir integridade e privacidade do corpo inteiro de uma mensagem RPC. Por esse motivo ele é chamado de um esquema de segurança (security flavour) ao invés de um esquema de autenticação (authentication flavor) como é chamado o tradicional AUTH_SYS. Uma implementação compatível com o NFSv4 obrigatoriamente deve implementar dois mecanismos de segurança sob a interface GSS. São eles: Kerberos V5 e LIPKEY.

2.2.2 Kerberos Version 5

O Kerberos (http://web.mit.edu/kerberos/www/) é um protocolo de autenticação que faz uso apenas de criptografia simétrica e consegue garantir o mesmo nível de segurança dos algoritmos de chave pública. Maiores detalhes sobre o funcionamento interno desse protocolo podem ser encontrados em [Koh93].

O protocolo Kerberos funciona muito bem para redes locais, mas não se aplica no caso da Internet pois exige controle centralizado. Cada domínio deve possuir um KDC (Key Distribution Center), que é o servidor de autenticação que irá fornecer os tickets para os clientes acessarem os serviços dentro do domínio. O KDC autentica os usuários e a aplicação que fornece o serviço. Para isso ele deve possuir uma base de dados com as chaves secretas de todos os usuário e serviços. Como normalmente os clientes NFS são implementados em nível de kernel, os tickets ficam armazenados no kernel e, toda vez que um usuário desejar acessar o serviço, seu respectivo ticket será utilizado.

O mecanismo do Kerberos usado na GSS-API especifica 3 modos de garantir integridade. No NFSv4 foi escolhido o algoritmo DES/MAC/MD5 que consiste em computar o DES-CBC MAC em paralelo com o MD5. Desse modo o NFSv4 consegue garantir robustamente a integridade das mensagens.

Quanto à privacidade, o mecanismo Kerberos V5 GSS_API prevê a utilização do DES (56 bits). A própria RFC do NFSv4 alerta para o fato deste algoritmo não ser robusto a ataques de força bruta nos dias atuais. Portanto, o Kerberos v5 no NFS provê uma garantia fraca quanto à privacidade dos dados.

O Kerberos não autentica o host servidor, mas apenas a aplicação que oferece o serviço. Porém isso não é preocupante, pois, para impersonar o serviço em outro host, o atacante precisaria possuir a chave secreta do serviço legítimo para assim poder decifrar o ticket e obter a chave de sessão. Como a chave secreta do serviço é um segredo de responsabilidade do administrador do sistema, pode-se considerar pouco relevante esse problema.

Um ponto fraco do Kerberos para sua utilização no NFS é o fato dele autenticar apenas o usuário e não autenticar o host cliente. As implementações de NFS atualmente possuem filtros para selecionar quais máquinas terão acesso aos sistemas de arquivos exportados. Esses filtros continuarão baseando-se apenas no endereço IP para autenticar o host cliente. Portanto, o uso de Kerberos não resolve o problema da falta de autenticação do cliente apontado no item 3.3.3. O administrador não teria como garantir, por exemplo, que

certas máquinas não confiáveis (com possíveis vírus, segurança frágil, máquinas de acesso público) sejam impossibilitadas de acessar o serviço NFS.

2.2.3 LIPKEY

O LIPKEY (Low Infrastructure Public Key) [Eis00] é um outro mecanismo de segurança que deve ser fornecido numa implementação NFSv4-compatível. Este mecanismo provê um modelo e segurança equivalente ao SSL (Secure Socket Layer) para utilização do NFS através da Internet.

O cliente recebe o certificado do servidor e compara com uma lista de autoridades certificadoras pré-definida. Caso o certificado tenha sido assinado por uma autoridade confiável então o serviço é considerado legítimo. Então o cliente envia a senha do usuário e uma chave de sessão simétrica (gerada aleatoriamente), ambas cifradas com a chave pública do servidor. O servidor pode então autenticar o usuário baseado em sua senha de acesso ao serviço. Pode também garantir privacidade dos dados, pois a partir dessa etapa todos os dados serão cifrados através da chave simétrica fornecida.

O NFSv4 obriga a implementação do algoritmo cast5CBC para garantir a confidencialidade, embora seja possível negociar outros algoritmos. Este algoritmo é considerado bastante robusto para o estado da arte atual portanto no aspecto da privacidade o LIPKEY é bastante adequado.

Quanto à integridade dos dados, a especificação do NFSv4 permite que sejam negociados diversos algoritmos, mas requer que o algoritmo HMAC-MD5 esteja sempre disponível. Este também é um algoritmo considerado robusto para garantir integridade dos dados, portanto nesse aspecto o LIPKEY é seguro.

O LIPKEY apresenta o mesmo problema de autenticação que o Kerberos levantado anteriormente. O LIPKEY autentica apenas o usuário e o serviço (do lado do servidor), mas as máquinas cliente e servidora não o são.

2.2.4 Negociação da Segurança

Tendo em vista que o NFSv4 pode oferecer diversos mecanismos de segurança, o cliente precisa de algum método para negociar qual o mecanismo de segurança a ser usado na comunicação com o servidor. Um servidor que exporta diversos pontos de montagem pode

especificar quais os algoritmos adequados para cada uma dessas entradas.

Para realizar essa negociação, foi introduzida uma nova operação no NFS chamada SECINFO, que permite que o cliente pergunte qual a segurança que o servidor requer para acessar determinado diretório. Os argumentos e resultados dessa operação devem ser segurados utilizando um dos esquemas de segurança mandatórios. Isso é para que o atacante não consiga interceptar a negociação e forçar o uso de níveis mais baixos de segurança.

A operação SECINFO retorna a lista (ordenada por prioridades) de esquemas de segurança suportados pelo servidor e também, dependendo do esquema, qualquer informação adicional necessária. Por exemplo, no caso do RPCSEC_GSS o servidor e o cliente precisam negociar o mecanismo a ser utilizado e também se será usado integridade ou privacidade.

A partir da resposta do SECINFO o cliente escolhe um esquema dentro da lista dos possíveis e inicia a execução do procedimento remoto. Note que o NFSv4 obriga que toda implementação NFSv4-compatível suporte determinados algoritmos de criptografia forte e que os mesmo tenham prioridade sobre os algoritmos de baixa ou nenhuma segurança. Portanto, o cliente não poderá escolher um esquema inseguro, como o AUTH_SYS pode exemplo, a mesmo que o servidor só permita este, pois haverá sempre um esquema mais seguro com maior prioridade disponível no servidor.

2.2.5 Controle de Acesso

No NFSv4 os atributos de um arquivo são variáveis. Existem atributos mandatórios e recomendados. Dentre os atributos recomendados há quatro que são referentes ao controle de acesso. São eles: User, Group, Unix mode bits e ACL.

Os três primeiros atributos são os atributos já existentes nas versões antigas do NFS. Com a diferença que agora o usuário e o grupo são definidos por uma string e não por um inteiro. Isso implica no fato de que o mapeamento de UID e GID para nomes não precisa ser o mesmo nas máquinas cliente e servidor.

O último atributo refere-se à utilização de ACLs - Access Control Lists (Listas de Controle de Acesso). Uma ACL é simplesmente uma lista que descreve quais os usuários e grupos que têm acesso a determinados serviços, sob certas restrições. O NFS versão 4 introduz suporte a ACL baseado no modelo do Windows NT e não no padrão POSIX,

pois o modelo das ACLs do NT é muito mais completo e mais difundido.

Cada entrada na ACL é chamada de ACE (Access Control Entry) e pode definir quatro ações diferentes: ALLOW, DENY, AUDIT, or ALARM. As ações ALLOW e DENY permitem ou negam respectivamente o acesso ao arquivo e são suportadas pela POSIX ACL. A operação AUDIT faz com que seja gerado um log da tentativa de acesso ao objeto. ALARM um alarme no sistema caso seja feito um acesso a esse objeto.

Existe um grande diferença entre o modelo NT e o POSIX que representa uma potencial incompatibilidade entre os dois modelos. No modelo NT, a primeira ACE que permite (ou barra) o acesso correspondente a um usuário ou grupo decide se o acesso será permitido ou negado. No modelo POSIX existem dois tipos de ACLs: entradas de usuário e entradas de grupo. Primeiramente são checadas todas as entradas de usuários e posteriormente a dos grupos. Porém todas as ACE´s são verificadas. Por essa e outras diferenças, a compatibilidade dos sistemas que utilizam POSIX ACL com o NFSv4 será problemática. Por esse motivo, já está sendo desenvolvida uma RFC exclusiva para especificar como deve ser feito o mapeamento de ACLs entre POSIX e NFSv4.

Portanto, o mecanismo de ACLs do NFSv4 é um método bastante adequado para controle de acesso porém a utilização deste recurso é problemática, tendo em vista que os diferentes sistemas operacionais adotaram padrões diferentes e incompatíveis de ACLs. Para facilitar essa situação está sendo desenvolvida uma RFC especificamente para tratar deste problema de compatibilidade.

2.2.6 Outros aspectos de segurança

O NFSv4 eliminou a necessidade do portmapper para descobrir a porta onde o servidor aguarda conexões. O novo protocolo fixa a porta 2049 como a porta padrão do NFS versão 4. Essa mudança, aliada ao uso do TCP, favorece o uso do NFS na Internet e facilita a filtragem desse protocolo através de firewalls.

2.2.7 Considerações finais sobre o NFSv4

O NFSv4, sob o ponto de vista da segurança, representa um avanço extraordinário em relação às versões anteriores. Há mecanismos para garantir autenticação, integridade e privacidade, de forma bastante robusta. Apesar de não haver autenticação da máquina

cliente e servidora, isso não é tão preocupante, pois há a autenticação do usuário e da aplicação servidora. Dessa forma é possível que um usuário legítimo obtenha acesso a partir de uma máquina não permitida, mas isso pode ser impedido com o uso de filtros de pacotes. Há avanços significativos também no aspecto de controle de acesso aos arquivos utilizando ACLs, porém a ampla compatibilidade desse recurso ainda é duvidosa.

Um aspecto bastante negativo do NFSv4 é a sua complexidade. A especificação do protocolo é bastante extensa e incompleta e, além disso, muitas informações são referenciadas de outras RFCs, como, por exemplo, informações sobre os mecanismos e protocolos de segurança. Diversos assuntos como replicação e migração de servidores estão em aberto. Muitos algoritmos e mecanismos de segurança são especificados como mandatórios para que a implementação seja considerada NFSv4-compatível. A grande complexidade do protocolo pode dificultar bastante a sua ampla utilização, tendo em vista que torna as implementações muito caras e demoradas.

Capítulo 3

Análise de Trabalhos Correlatos

Tendo em vista as deficiências do NFSv3 e a possibilidade de novos recursos não explorados pelo NFS, vários outros sistemas de arquivos de rede foram desenvolvidos. Alguns deles utilizaram o NFS como base, já outros optaram por desenvolver sistemas alternativos ao NFS.

No item 3.1 serão apresentados os trabalhos realizados que utilizam o NFS como ponto de partida ou como sistema base em sua implementação. Em seguida, no item 3.2, apresentaremos os sistemas de arquivos de rede alternativos ao NFS, que não utilizam o NFS como base.

3.1 Variantes do NFS

Devido a sua estabilidade e sua ampla utilização, o NFS (v2 e v3) serviu de base para diversos outros protocolos. Ao invés de projetar um novo protocolo, optou-se por estender o NFS para prover novas funcionalidades. Nesta seção iremos apresentar um resumo de alguns destes protocolos.

3.1.1 Trusted NFS (TNFS)

O TNFS é uma extensão do protocolo NFSv2 que suporta controle de acesso a arquivos baseado em níveis de segurança (MLS - *Multilevel Secure*). Esse tipo de controle de acesso é conhecido como MAC (*Mandatory Access Control*). O TNFS também oferece controle de acesso discricionário (DAC) através de listas de acesso.

O TNFS foi desenvolvido para suportar os requisitos da TCSEC (Trusted Computer System Evaluation Criteria) _ um modelo baseado em níveis de segurança especificado pela Agência de Inteligência da Defesa dos Estados Unidos. O TCSEC traz um conjunto de requisitos de segurança para sistemas computacionais que lidam com informações classificadas. Estes requisitos são descritos por [Woo87]. O TNFS foi projetado para suportar segurança em nível B1, que atribui rótulos de segurança para pessoas e documentos. O rótulo de segurança informa o nível de segurança de um documento ou de uma pessoa. Uma pessoa não pode acessar informações de níveis de segurança superiores ao seu próprio nível.

Para implementar esses requisitos de segurança, o TNFS definiu um novo esquema de segurança (*security flavour*) chamado AUTH_MLS. Através deste esquema todas as chamadas RPC são identificadas com as informações do nível de segurança do usuário. Através dessas credenciais o servidor pode utilizar suas políticas de controle de acesso para atribuir as permissões adequadas. Porém, como não há nenhum tipo de mecanismo criptográfico, o AUTH_MLS sofre as mesmas falhas de segurança do AUTH_SYS tradicional. É possível forjar requisições com credenciais de terceiros e não há mecanismos que permitam que o servidor detecte isso.

O TNFS foi desenvolvido pelo grupo *Trusted Systems Interoperability Group* que envolvia representantes de diversas empresas como DEC, AT&T, Apple, Sun, SecureWare e Silicon Graphics. Uma versão rascunho (draft) da especificação deste protocolo chegou a ser publicada, mas a especificação final não chegou a ser concluída.

3.1.2 WebNFS

O WebNFS ([Cal96a] e [Cal96b]) foi uma tentativa de adaptar o NFS para que ele pudesse ser usado na internet como um protocolo de transferência de arquivos da mesma forma que o HTTP e o FTP são usados. O WebNFS traz diversas modificações que foram aproveitadas posteriormente no NFSv4 porém na questão da segurança a única novidade em relação ao NFSv3 é a introdução de um protocolo de negociação do esquema de segurança bastante simples [CEC00]. Essa negociação é feita através de uma modificação no procedimento LOOKUP do NFS, permitindo que o cliente consulte a lista de esquemas suportados pelo servidor. Os esquemas de segurança utilizados são os mesmos do NFSv3, portanto, essa funcionalidade não representa qualquer melhoria significativa da segurança

do NFS.

Existe ainda uma outra modificação introduzida pelo WebNFS que está indiretamente ligada à segurança que é a utilização de uma porta fixa (porta 2049) usando o protocolo TCP. Em protocolos baseados em UDP, devido à inexistência de confirmações e números de sequência, torna-se mais fácil para um atacante enviar datagramas maliciosos e/ou repetir datagramas previamente gravados. A utilização de uma porta fixa também contribui para a segurança pois elimina a necessidade do *portmapper*, tornando a integração com firewalls mais fácil e evitando possíveis ataques ao próprio serviço *portmapper*.

O WebNFS não obteve ampla adoção pela comunidade, sendo muito pouco conhecido e utilizado. Existe apenas uma implementação em Java desenvolvida pela *Sun Microsystems*.

3.1.3 SNFS

O SNFS (http://www.math.ualberta.ca/imaging/snfs/) é um sistema capaz de redirecionar o tráfego do NFS para um túnel SSH. No caso do NFS sobre TCP, o redirecionamento pode ser feito diretamente pelo SSH, porém no caso do NFS sobre UDP isso não seria possível. Isso se deve ao fato do SSH somente estabelecer túneis TCP. Como o NFS sobre UDP é o padrão em todos os Unix, é necessário adicionar uma camada de software para realizar as modificações necessárias.

O SNFS implementa um redirecionador específico de mensagens RPC. Este programa é responsável por tunelar os pacotes UDP para uma conexão TCP que o SSH poderá redirecionar. Nenhuma alteração no código do cliente ou do servidor NFS é necessária. A instalação do SNFS se dá apenas através da instalação dos redirecionadores e de uma configuração específica.

No servidor, basta instalar o pacote do SNFS e configurar o arquivo /etc/exports para exportar os diretório desejados para a máquina localhost.

```
# /etc/exports
/mnt/mp3 localhost(async,rw)
/mnt/work localhost(1sync,rw)
```

Na máquina cliente é necessário rodar um script de configuração do SNFS:

```
# snfshost REMOTE:MOUNTPROG
```

Onde REMOTE é o nome da máquina servidora e MOUNTPROG deve ser um número de programa disponível no RPC. Em seguida, é necessário editar o arquivo /etc/fstab para estabelecer as opções de montagem.

```
# /etc/fstab
LOCAL.DOMAIN:/DIR /REMOTE/DIR nfs user,noauto,hard,intr,rsize=8192,
wsize=8192,moutprog=MOUNTPROG, nfsprog=NFSPROG00
```

Em seguida é preciso criar o diretório local de montagem e inicializar o redirecionador.

```
# smkdirall
# rpc_psrv -r -d /usr/local/etc/snfs/REMOTE
```

O usuário será solicitado para digitar a senha de root do servidor. A partir desse momento, o usuário pode realizar montagens normalmente.

```
# mount /REMOTE/DIR
```

A grande limitação do sNFS é a necessidade de se conhecer a senha de root do servidor. Isso restringe drasticamente os cenários de utilização deste serviço. Além disso representar uma restrição de uso, esse fato representa um ponto negativo para a segurança. A conexão ao servidor SSH é feita usando as credenciais do usuário root, portanto o servidor SSH precisar estar habilitado para aceitar logins deste usuário, e isso não é aconselhável como política de segurança.

O sNFS é capaz de garantir a privacidade e a integridade dos dados de forma confiável. Porém a autenticação continua sendo tão fraca quanto ao AUTH_SYS convencional. Apenas da conexão SSH ser autenticada, as mensagens RPC que estão sendo tuneladas não passam por nenhum processo adicional de autenticação. As mensagens chegam ao servidor NFS que simplesmente confia nas informações de credenciais contidas nas mensagens RPC. Portanto, qualquer usuário pode enviar mensagens RPC com UID (*User Id*) forjado ou mesmo utilizar o comando su para impersonar qualquer outro usuário.

Tendo em vista as grandes deficiências de segurança do SNFS e da relativa dificuldade

de implantação, o SNFS não é recomendado. Para muitos cenários, uma opção similar porém mais simples e segura é o SHFS que será abordado neste capítulo.

3.1.4 CryptosFS

O CryptosFS [O'S00] é sistema de arquivos distribuídos baseado no NFS, desenvolvido por Declan O'Shanahan. Este sistema faz uso de criptografia para garantir autenticidade, integridade e confidencialidade. A implementação consiste em uma modificação aplicada ao cliente e servidor NFS, sendo necessária a recompilação do kernel para a instalação do sistema, o que dificulta a instalação do sistema. Apesar de utilizar o NFS como base, a abordagem do CryptosFS é bastante diferente do NFS pois os arquivos são armazenados de forma cifrada no servidor. É utilizada criptografia simétrica (Blowfish) para cifrar arquivos e meta-dados com a finalidade de prover confidencialidade. O CryptosFS também faz uso de criptografia assimétrica para criar assinaturas digitais. A validação dessas assinaturas proporciona autenticidade e integridade. O servidor NFS possui a chave pública de cada arquivo permitindo verificar requisições de leitura ou escrita recebidas do cliente.

O CryptosFS substitui os mecanismos de controle de acesso do NFS por um mecanismo criptográfico de controle de acesso utilizando criptografia assimétrica. Quando um cliente deseja realizar uma operação de leitura, ele envia junto com a requisição uma assinatura digital. O servidor valida as requisições de leitura usando a chave pública do arquivo para checar a assinatura. Como os arquivos são armazenados já na forma cifrada o servidor não precisa cifrar os dados, eles são simplesmente repassados para o cliente caso a validação tenha sido bem sucedida. Para realizar escritas, o cliente precisa possuir a chave privada do arquivo. O servidor valida a requisição decifrando a assinatura com a chave pública do arquivo.

O ciframento dos dados é sempre realizado pelo cliente. Isso reduz pela metade o número de operações criptográficas. Em outros sistemas, os dados são cifrados pelo cliente e, em seguida, decifrados pelo servidor para armazenamento, ou vice-versa. Outra vantagem é que, pelo fato dos arquivos estarem cifrados no servidor, nem mesmo o administrador conseguiria ter acesso aos conteúdos desses arquivos. Por outro lado, em alguns cenários isso pode representar uma desvantagem. Por exemplo, em laboratórios escolares ou em algumas corporações, muitas vezes é desejado que pessoas de cargos mais altos tenham acesso de leitura a todos os documentos para fins de monitoração. Outra

grande desvantagem dessa abordagem é que ela é incompatível com a maioria dos sistemas de arquivos distribuídos no sentido de que não seria possível montar um servidor multi-protocolos que servisse CryptosFS e NFS ao mesmo tempo por exemplo.

O CryptosFS não possui uma implementação de produção. Há somente uma implementação protótipo utilizada para prova de conceito. O CryptosFS apesar de ter características de segurança bastante interessantes dificilmente seria um sistema amplamente utilizado devido às desvantagens e inconveniências apontadas aqui.

3.2 Outros Sistemas de Arquivos Distribuídos

3.2.1 Andrew File System - AFS

O Andrew File System [Mor86] é um projeto desenvolvido pela Universidade de Carnegie-Mellon desde 1983. Seu objetivo era prover um sistema de arquivos distribuído escalável que pudesse prover compartilhamento de arquivos à milhares de máquinas em um campus de universidade.

Essa escalabilidade do AFS é atingida através do uso agressivo de mecanismos de cache nos clientes, que consequentemente interagem menos com o servidor. Além disso, passou-se grande parte das tarefas dos servidores para os clientes. Nas primeiras versões do AFS, quando um arquivo era aberto, todo o arquivo era baixado do servidor para o cliente, porém mais tarde começou-se a trabalhar com blocos de 64KB. Todas as operações de leitura e escrita são efetuadas na cópia local do arquivo, e ele somente será transferido de volta para o servidor quando o arquivo for fechado (desde que ele tenha sido modificado). Uma conseqüência deste mecanismo é que o AFS faz uso de semântica de sessão, ou seja, um usuário A só poderá visualizar uma alteração de um arquivo realizada por um outro usuário B quando A abrir o arquivo depois que B já o tiver fechado.

Ao lado da escalabilidade, um outro problema importante que os desenvolvedores decidiram enfrentar era a segurança. Esse problema era potencializado pelo fato do AFS ter sido projetado para um cenário com uma quantidade grande de clientes e usuários. A solução adotada foi a utilização do protocolo Kerberos para autenticação mútua entre clientes e servidores.

O Kerberos [Ste88] é um serviço de autenticação distribuída desenvolvido no MIT (Massachusetts Institute of Technology) que permite um parceiro provar sua identidade

perante um outro parceiro sem a necessidade de enviar dados confidenciais pela rede. Esse processo é realizado como um serviço de autenticação que envolve um terceiro parceiro confiável utilizando apenas criptografia simétrica. Opcionalmente ele também fornece integridade e confidencialidade das mensagens transmitidas. Ao logar-se no AFS o usuário digita sua senha e recebe do servidor Kerberos um TGT *Ticket Granting Ticket*, que é um ticket que prova sua identidade perante o TGS (*Ticket Granting Service*). O TGS é o serviço que finalmente irá fornecer os tickets individuais para acessar todos os serviços da rede incluindo o AFS. O motivo de haver esse ticket intermediário é um dos pontos-chave do sucesso do Kerberos. O usuário digita sua senha pessoal e o sistema, a partir dela, obtém um TGT junto ao servidor de autenticação. A partir daí o sistema não precisa mais da senha do usuário e pode apagá-la da memória, pois com o TGT em mãos a máquina pode obter futuramente todos os tickets para acessar quaisquer outros serviços da rede. O TGT possui uma validade portanto mesmo que o TGT seja comprometido, a utilização deste ticket será limitada.

O AFS traz a sua própria versão de Kerberos baseando-se na versão 4 do Kerberos com algumas melhorias, porém é possível (e recomendável) configurar o AFS para utilizar um servidor Kerberos 5. A versão 4 do Kerberos apresenta um nível de segurança bastante razoável para a maioria dos cenários, mas existem algumas falhas apontadas por Bellovin e Merrit em [SB91]. O Kerberos 5, além de corrigir essas falhas, apresentam novos recursos e contornam algumas limitações da versão antiga. Além disso é aconselhável não utilizar o Kerberos do AFS pois seu uso é restrito ao AFS e outros serviços não irão poder utilizar os benefícios do Kerberos.

O AFS tornou-se um produto de uma empresa chamada Transarc ligada às pessoas que desenvolveram o AFS na Universidade de Carneggie-Mellon. Posteriormente, a IBM adquiriu a Transarc. E em 2001, a IBM decidiu bifurcar o seu produto AFS em dois projetos, uma versão proprietária e outra versão de código aberto. Esta última, chamada OpenAFS (http://www.openafs.org), é mantida e suportada pela comunidade de software livre. Trata-se de um código bastante testado e portanto estável, disponível para quase todas as plataformas, entre elas: Linux, AIX, FreeBSD, OpenBSD, Irix, HP/UX, MacOS, Solaris e Windows. O OpenAFS é o maior concorrente do NFS atualmente e para a grande parte dos cenários é a alternativa mais indicada do ponto de vista da segurança.

3.2.2 Coda

O Coda [Sat90] é um projeto extremamente ambicioso desenvolvido no início dos anos noventa na Universidade de Carneggie-Mellon pelo Professor Satyanarayanan. O Coda é um descendente do AFS e seu foco principal é prover acesso a sistemas de arquivos distribuídos para computadores móveis provendo operação desconectada.

A parte mais interessante do Coda é a possibilidade de continuar acessando um sistema de arquivos distribuído mesmo após se desconectar da rede. Se um arquivo está armazenado localmente na máquina, o usuário pode ler e escrever no arquivo sem a prévia permissão do servidor. Quando esse computador for reconectado na rede, o sistema tenta propagar as alterações realizadas nos arquivos enquanto desconectado para os servidores apropriados. Conflitos podem ocorrer (por exemplo, dois clientes podem ter alterado o mesmo arquivo enquanto estavam desconectados). O Coda provê ferramentas para o usuário decidir qual cópia deve prevalecer.

Com o objetivo de poder oferecer acesso a arquivos enquanto desconectado, a parte cliente do Coda tenta armazenar a maior quantidade possível de arquivos no disco local, para o caso da conexão com o servidor ser perdida. Isto é realizado por um mecanismo que associa prioridades para cada arquivo baseado na última vez que o arquivo foi acessado, e uma lista de arquivos importantes é fornecida pelo usuário (chamada de *hoard database*), para deixá-los disponíveis mesmo estando offline. A cada 10 minutos, um processo chamado hoard walk é iniciado para trazer para o disco local todos os arquivos com prioridade mais elevada. A idéia é que todo o trabalho do cliente esteja armazenado no disco local. Se o disco local encher, e um novo arquivo precisar ser trazido, o arquivo com menor prioridade do disco é descartado, dando lugar para esse novo arquivo.

Podemos ver que o Coda não respeita a semântica de sessão (ao contrário do AFS), mas isso é tolerável considerando que queremos uma disponibilidade extra do sistema de arquivos.

O Coda também implementa mecanismos automáticos de replicação não presentes no AFS. Cada volume (um conjunto de diretórios do sistema de arquivos) é associado a um *volume storage group* (VSG), que consiste de um conjunto de servidores que replicam o volume. O conjunto de servidores acessíveis de um certo grupo em um certo momento é chamado de AVSG. A coerência entre as várias cópias de um arquivo é mantida por um sistema parecido com o de callbacks do AFS. Quando um cliente envia uma atualização

de um arquivo para o servidor, a atualização é enviada para todos os servidores AVSG usando um mecanismo multiRPC. Se um servidor que estava caído levanta, nada é feito inicialmente para atualizar seus arquivos. Porém, sempre que um cliente envia uma requisição para abrir um arquivo para o seu servidor preferido, ele também pede a todos os servidores AVSG que enviem a versão daquele arquivo que eles detêm. Assim, o cliente pode descobrir se existe algum servidor com uma cópia desatualizada, avisando-o para atualizar esse arquivo.

Do ponto de vista de segurança, o Coda não traz nenhuma modificação em relação ao seu precedente, o AFS. O Coda também utiliza o protocolo Kerberos para autenticação mútua entre clientes e servidores.

Embora o Coda seja teoricamente mais sofisticado que o AFS, o Coda é um projeto de pesquisa que nunca alcançou uma estabilidade para ser utilizado em ambientes de produção. Por outro lado, o AFS já está em produção em diversos lugares do mundo há anos. Portanto o Coda é uma opção desaconselhável exceto para pessoas que desejam experimentar e desenvolver sistemas de arquivos distribuídos.

3.2.3 Intermezzo

O Intermezzo [Hag02] também foi desenvolvido na Universidade de Carneggie-Mellon por Peter Braam. Este sistema foi profundamente inspirado no Coda porém foi totalmente reprojetado. Seu código não foi baseado no código do Coda, mas apenas as idéias e experiências foram aproveitadas. Trata-se de uma re-edição do Coda de uma maneira mais enxuta, leve e rápida, com a finalidade de torná-lo um produto. O objetivo do Intermezzo é prover as funcionalidades e técnicas desenvolvidas no AFS e no Coda, mas de forma simétrica, assíncrona, e com uma ordem de magnitude mais simples, visando prover alta disponibilidade.

Este sistema permite funcionalidades avançadas, como operação desconectada (permitindo que se trabalhe em arquivos que estão no cache local) e é apropriado para aplicações de alta disponibilidade, em que se precisa garantir que o armazenamento nunca estará indisponível (ou simulada, se estiver fora do ar). Ele também tem aplicações para manter os dados em sincronia entre múltiplos computadores, como um laptop ou PDA e um computador desktop. O Intermezzo possui implementação apenas para o Linux e foi incluído no kernel deste a partir da versão 2.4.5.

O Intermezzo consiste em um gerenciador de cache em nível de usuário (Lento) escrito em Perl, e um módulo de kernel (Presto). O Presto atua como um driver de filtro para as requisições à VFS das aplicações. Ele pré e pós-processa as requisições, gerenciando faltas no cache e mantendo um registro das modificações que serão enviadas para o servidor. O Presto funciona em conjunto com o gerenciador de cache em nível de usuário, cuja tarefa básica é transferir os dados através da rede. Para o funcionamento adequando, o Intermezzo precisar atuar no topo de um sistema de arquivos local com um *journal* como o ext3 ou reiserfs. Esse *journal* é utilizado pelo Intermezzo para informar as mudanças para outros computadores que estejam servindo os mesmos arquivos.

O suporte a operação desconectada é realizado mantendo um log, chamado KML (*Kernel Modification Log*), de todas as alterações que ocorram enquanto disconectado do servidor. O Intermezzo pode suportar POSIX ACLs ao invés do tradicional esquema usuário/grupo/outros, mas desde que seja aplicado um patch apropriado ao kernel.

No aspecto da segurança, o Intermezzo ainda não implementou nenhum subsistema de segurança apresentando um nível de segurança similar ao NFS com AUTH_SYS. O que, na prática, limita o uso do Intermezzo à redes fechadas e confiáveis. Segundo o autor, constam planos de integração com o Kerberos e também do uso de SSL para prover segurança.

O código base do Intermezzo é relativamente pequeno portanto isso diminui a quantidade de bugs no software, e consequentemente aumenta a estabilidade e a segurança do sistema, além de facilitar a portabilidade. Por outro lado, o Intermezzo é um sistema recente e seu código ainda não está suficientemente testado e estável. Apesar de não estar oficialmente inativo, o projeto não apresenta atividades há mais de dois anos. Portanto, para as pessoas que não necessitam das funcionalidades exclusivas do Intermezzo, é aconselhável não utilizá-lo em ambientes de produção.

3.2.4 SHFS

O shfs (http://shfs.sourceforge.net), cujo nome significa *(Secure) sHell File System*, é um sistema de arquivos remoto que utiliza o SSH para prover segurança. Trata-se de um modulo para o kernel do Linux que permite realizar a montagem de sistemas de arquivos através de uma conexão de SSH.

A figura 3.1 foi extraída da página oficial do shfs e mostra o funcionamento deste

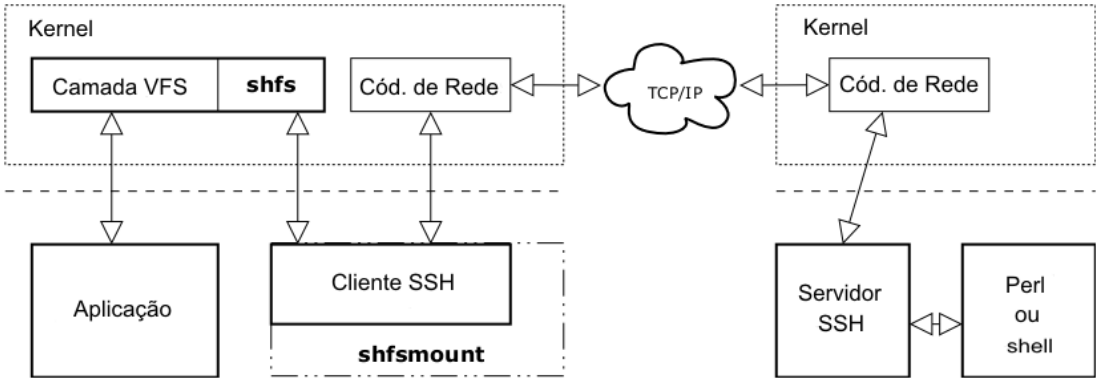

Figura 3.1: Diagrama do shfs

sistema. A aplicação realiza uma chamada mount ao sistema de arquivos. A camada VFS (Virtual File System) redireciona essa chamada para o módulo shfs e esse, por sua vez, utiliza um cliente SSH para enviar comandos para um servidor SSH. Do lado remoto (servidor) há apenas um servidor SSH convencional e uma coleção de programas (chmod, cut, dd, ln, ls, mkdir, touch, perl etc) que executam as operações requisitadas pelo cliente.

Esse sistema ainda apresenta um cache de arquivos para acelerar os acessos, porém ele não garante consistência quando múltiplos hosts acessam o mesmo arquivo. Os arquivos são copiados para o cache da máquina cliente e as modificações só são enviadas ao servidor ao fechar o arquivo.

Durante o processo de montagem o usuário é solicitado para entrar com sua senha. A autenticação também pode se dar através do mecanismo convencional de chaves públicas do SSH. A privacidade baseia-se na segurança provida pelo protocolo SSH através de algoritmos de criptografia simétrica usando uma chave de sessão estabelecida.

O shfs é um sistema extremamente fácil de ser instalado pois requer apenas a instalação de um módulo na máquina cliente. O lado do servidor não necessita de nenhuma instalação, bastando apenas ter um servidor de SSH previamente configurado. Mas como o SSH é um serviço absolutamente popular sendo a forma de terminal remoto padrão atualmente, todas as distribuições de Linux trazem o SSH já configurado e muitas vezes ativo por padrão. Dessa forma o shfs torna-se uma opção bastante atraente para o acesso remoto a um sistema de arquivos. Por esse motivo, várias distribuições como o Debian, Slackware e Kurumin, incluíram este programa em suas listas de pacotes.

Porém o shfs apresenta algumas desvantagens. A primeira delas diz respeito à portabilidade do sistema. Atualmente, só existe implementação do cliente para Linux e, devido ao fato de ser um módulo de kernel, sua portabilidade para outros sistemas operacionais é dificultada. Em uma rede heterogênea, com diferentes sistemas operacionais fica inviável utilizar o shfs para compartilhamento de arquivos. Outra desvantagem é que pelo fato de utilizar código em nível de usuário, a performance desse sistema é prejudicada devido às trocas de contexto entre o modo kernel e o modo usuário.

O shfs também apresenta o inconveniente de precisar que o usuário digite a senha no momento da montagem. Isso dificulta a utilização do autofs para realizar montagens sob demanda. No caso de uma queda de conexão do SSH, o usuário precisaria digitar novamente sua senha. Esse inconveniente pode ser contornado fazendo com que o shfs armazene a senha do usuário para utilizar em reconexões, porém essa medida não é indicada pois a senha do usuário fica exposta em claro na memória e poderia ser capturada por um atacante. A melhor maneira de contornar esse problema seria utilizar o mecanismo de autenticação por chave públicas do SSH.

Devido a esse problema, o shfs não é muito adequado para uso em uma rede interna como um sistema de arquivos centralizado seguindo o paradigma tradicional, onde os usuários utilizam uma área pessoal (homedir) que é montada automaticamente. Isso porque o mecanismo de chaves públicas do SSH busca a chave privada do usuário em seu homedir que ainda não está montado, e o qual não pode ser montado sem ter a chave privada.

O shfs é uma ótima alternativa para acesso remoto no caso de não haver uma VPN *(Virtual Private Network)*. Estabelecer uma VPN pode ser muito custoso quando se quer obter apenas um simples compartilhamento de arquivos. Configurar uma VPN requer um esforço maior e envolve restrições a serem contornadas.

3.2.5 DCE/DFS

O DFS é um aprimoramento baseado no AFS. Ele faz parte de um projeto maior da *Open Software Foundation* chamado DCE (*Distributed Computing Environment*) que é um padrão para um conjunto de tecnologias de computação distribuída.

O DCE ([Ros92] e [Fou93]) é um ambiente para aplicativos distribuídos rodando em diversas plataformas. Ele é utilizado especialmente em ambientes de hardware e software

heterogêneos, compostos de diferentes sistemas operacionais, para administrar os recursos de processamento disponibilizados para os aplicativos, bem como para controlar o acesso a esses recursos. O sistema dispõe ainda de serviços de segurança para proteger e controlar o acesso a dados, serviços de atribuição de nomes, que facilitam a procura por recursos distribuídos e um modelo de organização de contas de usuários, serviços, aplicativos e dados que estejam largamente distribuídos pelas máquinas da rede. Em 2005, o código-fonte do DCE foi aberto sob uma licença livre LGPL.

O DFS preserva o uso agressivo de cache no cliente para diminuir a carga sobre os servidores e a rede. Uma grande diferencial é que o DFS suporta a semântica *single-copy* através de um sistema de tokens, portanto contornando os problemas com a semântica de sessão.

A migração do AFS para o DFS tem sido muito lenta. Ambos os sistemas eram desenvolvidos e mantidos pela Transarc, que foi adquirida pela IBM. O DFS possui uma dependência de outros componentes do DCE como serviço de tempo e serviço de nomes e isso impõe um *overhead* grande na instalação em ambientes sem o DCE. Espera-se que com a abertura do código-fonte em 2005, o DCE se torne mais popular e o DFS passe a ser mais utilizado.

O DFS, assim como o Coda e o AFS, utiliza o Kerberos versão 5 como esquema de segurança, o que propicia um alto nível de segurança.

3.2.6 SMB/CIFS

O CIFS (*Common Internet File System* é uma simples renomeação do SMB (*Server Message Block*, um conjunto de protocolos desenvolvido para que PCs pudessem compartilhar recursos (arquivos, impressoras, portas seriais etc) em uma rede local.

O SMB foi criado por um conjunto de empresas entre elas a Microsoft, IBM e Intel no final dos anos 80, porém somente a Microsoft prosseguiu com o desenvolvimento sendo adotado como protocolo nativo para troca de arquivos no Microsoft Windows 95, Windows NT e OS/2. O SMB, ao longo do tempo, foi sendo aprimorado e modificado. Anos mais tarde, a Microsoft e um grupo de outras empresas (Digital, SCO, NetApp etc) decidiram desenvolver uma versão publica e aberta do SMB usando o nome CIFS. Atualmente, os termos SMB e CIFS se referem ao mesmo protocolo, porém a própria Microsoft enfatiza o uso do termo CIFS, apesar de algumas referências ao SMB ainda ocorrerem. Iremos

utilizar apenas o termo CIFS daqui a diante.

O CIFS opera de forma semelhante ao NFS, permitindo que programas e usuários manipulem arquivos remotos como se fossem arquivos locais. O protocolo CIFS também trabalha com chamada de procedimentos remotos. O cliente envia um pacote com a requisição para o servidor. Cada pacote é tipicamente uma requisição básica para uma operação como: leitura, abertura ou criação de um arquivo. O servidor recebe este pacote, checa se a requisição é legal, verifica se o cliente possui permissões apropriadas e finalmente executa a operação retornando o resultado para o cliente.

Embora, o foco principal do CIFS seja o compartilhamento de arquivos, há outras função que estão associadas a ele. Este serviço também é capaz de buscar outros servidores CIFS na rede (*browsing*), compartilhar impressoras e também realizar tarefas de autenticação.

Os clientes conectam-se ao servidor usando NetBIOS sobre TCP/IP (especificado na RFC1001 [Agg87a] e RFC1002 [Agg87b]), NetBEUI ou IPX/SPX como protocolo de transporte.

Existem implementações de CIFS para praticamente todas as plataformas. Devido ao grande número de usuário Windows tanto coorporativos como domésticos, foram desenvolvidas implementações para diversas plataformas. Todos os sabores de Unix dispõe do programa Samba (http://www.samba.org), que é uma implementação de cliente/servidor CIFS extremamente robusta e largamente utilizada em todo o mundo. As máquinas Apple também possuem implementações de cliente e servidor CIFS. Isso faz do CIFS o protocolo mais comum para compartilhamento de arquivos disponível no mercado.

O CIFS apresenta diversos esquemas de segurança que foram evoluindo com o passar do tempo.

Share-Level: Proteção em nível de compartilhamento

A proteção em nível de compartilhamento estabelece um modelo de segurança baseado em senhas para cada recurso compartilhado. Um administrador deseja compartilhar um recurso (diretório, impressora etc) deixando-o disponível na rede. Para protegê-lo, uma senha é atribuída a esse compartilhamento e somente o usuário que possuir essa senha conseguirá ter acesso ao recurso.

Esse mecanismo de segurança apresenta graves deficiências de segurança pois, em primeiro lugar, as senhas são transmitidas em claro pela rede. Qualquer atacante po-

deria capturá-la com certa facilidade especialmente em redes sem switches ou redes sem fio. Outro problema, é a falta de garantia não-repúdio. Como todos os usuários compartilham a mesma senha, não há como identificar as ações de cada usuário. Além disso, senhas de recursos são facilmente compartilhadas entre funcionários de uma empresa. Por outro lado, as pessoas resistem mais em compartilhar suas senhas pessoais. O fato de a senha ser compartilhada ainda gera um outro problema: quando um usuário deixa a empresa ele continua possuindo a senha, enquanto seria mais fácil remover um usuário.

User-Level: Proteção em nível de usuário

A proteção é aplicada para cada arquivo compartilhado e é baseada nas permissões do usuário. Os usuários possuem senhas individuais e cada usuário da máquina cliente deve ser autenticado pelo servidor. O cliente recebe um UID que deve ser apresentado em futuras requisições. A grande deficiência deste esquema é que novamente aqui a senha é transmitida em claro pela rede. Em 1997, o pacote de atualizações Service Pack 3 para o Windows NT 4.0 desabilitava esse tipo de autenticação por padrão, incentivando usuários a procurarem outros métodos.

LanMan 1.2 Desafio/Resposta

A especificação do CIFS sugere que essa forma de autenticação deveria ser usada somente por razões de compatibilidade reversa e recomenda o NT LM 0.12 como método preferível. Neste mecanismo a senha do usuário não é enviada em claro pela rede, porém o desafio inicial enviado pelo servidor é enviado em claro. Além disso, o algoritmo de hash usado era extremamente simples. Isso tornava trivial um ataque de força bruta principalmente pelo fato de que as senhas no LanMan não são sensitivas à maiúsculos portanto isso diminuiu bastante o número de combinações possíveis.

NT LM 0.12 Desafio/Resposta

Este é o método de autenticação introduzido pelo Windows NT e utilizado como padrão nas versões anteriores ao Windows 2000. O NTLM (*Windows NT Lan Manager*) permite senhas com letras maiúsculas e minúsculas e utiliza o MD4 para gerar o hash da senha do usuário e utiliza para cifragem o DES no modo de blocos.

O NTLM apresenta várias deficiências apontadas em [Bur04]. Uma das deficiências é

devida ao algoritmo DES que já não é considerado seguro para o estado da arte atual. Outro problema é a suscetibilidade do NTLM a ataques *man-in-the-middle*. Além disso, este método apresenta características que facilitam um ataque de dicionário ou mesmo por força bruta.

Em vista disso, o NTLM não é um método de autenticação indicado para sistemas com altos requisitos de segurança. Porém para a maioria das pequenas e médias empresas e redes domésticas, o NTLM provê um nível de segurança aceitável.

Visando contornar esses problemas, foi desenvolvido o NTLMv2, que é um aprimoramento da primeira versão. Este método corrige todas as falhas de seu antecessor, apresentando um alto nível de segurança. O NTLMv2 foi introduzido no *Service Pack* 4 do NT 4.0, e essa atualização é altamente recomendada.

Kerberos (Active Directory)

A introdução do *Active Directory* é uma das mudanças mais significativas nos recursos de rede do Windows. Essa tecnologia foi introduzida a partir do Windows 2000. O *Active Directory* é uma coleção de protocolos, incluindo o próprio CIFS e também outros como DNS, LDAP, Kerberos, DHCP e outros.

Há alguns problemas na utilização do *Active Directory* em ambientes Unix causados pelo fato da Microsoft adicionar extensões proprietárias a esses protocolos. Isso dificulta a interoperabilidade com outras implementações destes serviços, que seguem as especificações corretamente e não suportam as extensões do *Active Directory*.

Dentro do *Active Directory* a autenticação dos usuários é feita utilizando o Kerberos. Portanto, o CIFS apresenta o mesmo mecanismo de segurança presente no AFS e Coda. Como já vimos, esse esquema de segurança é capaz de prover um alto nível de segurança, logo ele deve ser usado preferencialmente em detrimento a todos os outros anteriores.

Capítulo 4

Solução Proposta - NFS Guard

A solução, batizada de NFS Guard, consiste em uma proposta em alto nível de uma arquitetura para agregar mecanismos de segurança ao NFSv3 que possam garantir privacidade, integridade e principalmente autenticidade das requisições. Na fase de projeto da solução a primeira atividade desenvolvida foi a definição dos requisitos desejados. A partir dos requisitos definidos foi elaborada uma solução que procurasse atender ao máximo esses requisitos. Essa foi uma tarefa trabalhosa devido à dificuldade de atender todos os requisitos definidos para a solução dentro do cenário do NFSv3, que é muito complexo e inseguro.

4.1 Especificação dos Requisitos

A primeira fase desta etapa do projeto consistiu na elaboração de uma lista de requisitos fundamentais desejados para a nossa solução. A partir dos requisitos definidos foi desenvolvida uma solução que procurasse atendê-los ao máximo. Os requisitos fundamentais considerados foram os seguintes.

Requisito 1: Preservar as funcionalidades oferecidas pelo NFSv3

Todas as funcionalidades oferecidas pelo NFSv3 devem ser preservadas o máximo possível, como por exemplo: opções de exportamento de hierarquias, restrições de máquina que podem acessar determinadas hierarquias, descritores de arquivos persistentes e suporte ao Autofs e NLM (*Network Lock Manager*). O objetivo desse

requisito é diminuir os impactos e os efeitos colaterais da implantação de nossa solução.

Requisito 2: Grande facilidade de implantação e administração do sistema

O sistema deve ser extremamente simples de ser instalado e administrado. Sistemas com alta complexidade de instalação, em geral, têm grande rejeição pelos administradores de sistemas. Idealmente, a instalação do sistema deve consistir no simples ato de instalar um pacote e ativar o programa através de um *script*. Será evitada ao máximo a necessidade de compilações personalizadas e configuração de outros serviços. Sempre que possível será dada a preferência por alternativas que não necessitem de configuração ou intervenção manual.

Requisito 3: Compatível com diferentes versões de kernel do Linux

A solução não deve ser restrita a versões muito específicas de kernel. Isso dificulta a distribuição do software e complica a tarefa do administrador em futuras atualizações do sistema operacional. É admissível que haja uma restrição quanto à versão majoritária do kernel. Por exemplo, é aceitável que a implementação só funcione no kernel 2.4, e para o kernel 2.2 ou 2.6 sejam necessárias algumas modificações e recompilações. O que deseja-se evitar é uma solução que tenha uma exigência muito restrita de kernel, por exemplo, kernel 2.4.5-18 pois nesse caso será necessário desenvolver e testar a implementação para cada uma das versões suportadas.

Requisito 4: Fácil portabilidade para outras plataformas

O nosso ideal é desenvolver um sistema que tenha condições de ser amplamente adotado no mercado. As redes em produção são em geral bastante heterogêneas, contando com diversas plataformas. Portanto, apesar da solução estar sendo desenvolvida para o Linux, tentou-se sempre procurar adotar tecnologias e mecanismos possíveis de serem portados para outras plataformas.

Requisito 5: Máxima transparência para o usuário

O NFSv3 provê um nível de transparência quase total para o usuário. O usuário acessa seus arquivos remotos como se estivessem armazenados localmente. Na nossa solução procurar-se-á manter esta transparência ao máximo, de modo que o cliente não precisará se comportar de maneira diferente para acessar o NFS através de nosso sistema de segurança.

Requisito 6: Autenticação do usuário é obrigatória

Esse é o maior ponto fraco do NFSv3 utilizando AUTH_SYS. Não há nenhum mecanismo de autenticação confiável do usuário que está fazendo a requisição. O nosso sistema deverá autenticar o usuário através da senha pessoal dele utilizando algum método criptográfico que garanta essa autenticidade.

Requisito 7: Autenticação do servidor é obrigatória

É necessário que o cliente tenha certeza de que está se conectando a um servidor legítimo e não a um software malicioso que se faz passar pelo servidor real. Caso contrário, o cliente acessará arquivos corrompidos e/ou enviará seus arquivos para o atacante. Para tanto é necessário que haja algum mecanismo criptográfico que autentique o servidor NFS perante o cliente.

Requisito 8: Autenticação da máquina cliente é desejável

A autenticação da máquina cliente no NFSv3 é feita somente através do endereçamento IP, o que é um método muito fraco pois é vulnerável a ataques de *IP Spoofing* [Nak03]. Porém, há cenários onde a autenticação do *host* cliente não é necessária. Em laboratórios, por exemplo, os usuários podem acessar seus arquivos a partir de qualquer máquina do laboratório, portanto não é necessário autenticar a máquina de origem que o usuários está utilizando. Desse modo, este requisito será considerado como opcional na configuração do sistema.

Requisito 9: Privacidade e Integridade dos dados obrigatória

O sistema não deve permitir nenhum tráfego em claro pela rede e deve garantir que nenhuma mensagem seja alterada por terceiros. Portanto, a privacidade e a integridade dos dados serão obrigatoriamente garantidas por meio de mecanismos criptográficos..

Requisito 10: Facilitar o tráfego de NFS através de Firewalls

O NFSv3 em geral utiliza o protocolo UDP como transporte. Esse protocolo dificulta a criação de regras para firewalls pois não há estabelecimento de conexão e além disso os números das portas utilizadas não são fixos. Ademais existe a presença de outros protocolos (Mount, Portmap, NLM e Statd) que são necessários para o funcionamento do NFS e também precisariam ser levado em conta na criação das

regras do firewall. Neste projeto será buscada uma solução que facilite o tráfego do NFS através de firewalls de maneira mais segura.

4.2 Arquitetura da Solução

Decidiu-se buscar uma solução que não necessitasse de qualquer alteração nos programas que compõem o NFSv3 que na sua maioria são implementados dentro do kernel. Se houvessem alterações no kernel do Linux, os administradores que quisessem utilizar nosso sistema teriam que recompilar o kernel com o patch contendo as alterações. Isso é uma tarefa de maior complexidade que deseja-se evitar para atender ao requisito 2. A necessidade de um patch de kernel para a instalação do sistema tornaria a solução dependente de versões específicas de kernel, o que não é desejado no requisito 3. Outro motivo para essa decisão seria a compatilibidade com sistemas de código fechado. Se o código do sistema não estiver disponível não há como efetuar as alterações necessárias para a implementação da solução, o que não atenderia o requisito 4.

Para atender aos requisitos 1 e 10, optou-se por uma arquitetura que estabeleça um túnel seguro entre o cliente e o servidor. Todo tráfego NFS, incluindo seus protocolos auxiliares, será desviado para dentro desse túnel. Como não há nenhuma alteração no código do NFS todas as funcionalidades já implementadas no NFSv3 serão mantidas o que atende ao requisito 1. O túnel entre os dois proxies utilizará o TCP como protocolo de transporte pois é um protocolo que tem uma interação muito mais fácil com firewalls. Além disso, o TCP é um protocolo que já oferece diversas funcionalidades como garantia de entrega, ordenação de pacotes, controle de fluxo e congestionamento, dentre outras. Caso se tivesse optado pelo UDP, essas funcionalidades teriam de ser implementadas no nível da aplicação, o que representaria mais uma complexidade para o projeto.

Outro ponto importante foi a opção por utilizar-se cliente e servidores NFS usando o UDP. Essa decisão foi tomada basicamente por dois motivos. Em primeiro lugar, por motivo de compatibilidade. Todas as implementações de NFS são compatíveis com o UDP e o mesmo não acontece com o TCP que possui poucas implementações estáveis. O segundo motivo é o fato do UDP ser um protocolo bem mais simples e fácil de ser processado. Com o UDP, as mensagens RPC estão contidas dentro de um datagrama. Caso se usasse o TCP, seria necessário lidar com a remontagem e retransmissão de segmentos para se

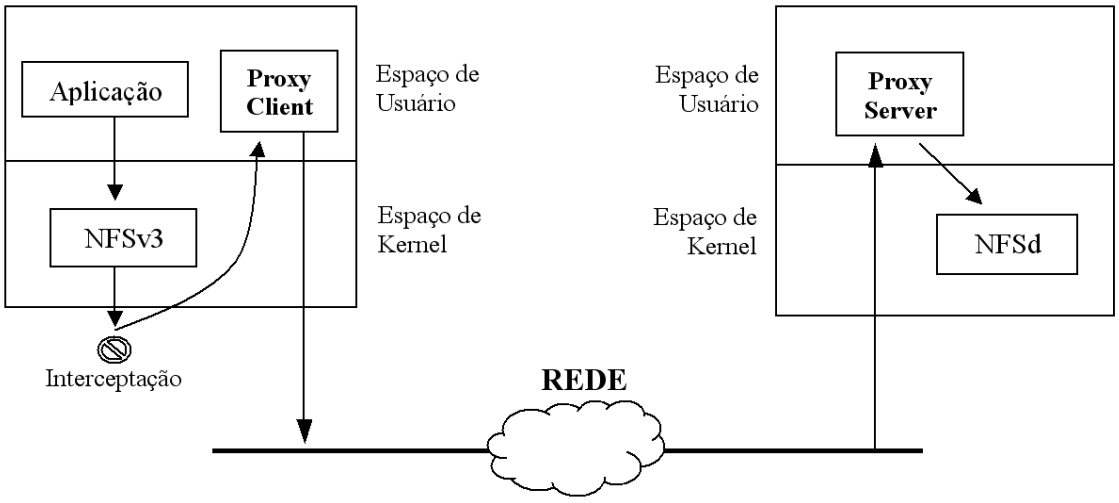

Figura 4.1: Arquitetura da Solução de Segurança para o NFSv3

recuperar uma mensagem.

A figura 4.1 mostra a arquitetura que é proposta nessa etapa do trabalho. A solução está baseada em dois componentes de software: o proxy client e o proxy server. Estes dois componentes são programas de nível de usuário que serão responsáveis por intermediar de maneira segura a comunicação entre o cliente e o servidor de NFSv3. Estes dois últimos estarão intactos, pois não sofrerão nenhuma alteração de código ou de configuração. Tanto o cliente quanto o servidor NFSv3 não terão conhecimento da existência desses proxies. A intermediação do proxy cliente é feita através da interceptação das requisições em nível de rede utilizando filtros de pacotes. No caso do Linux, é possível utilizar regras do iptables para desviar as requisições destinadas ao servidor NFS para o proxy local. Todas as plataformas em produção atualmente suportam esse recurso, que é fundamental para uma técnica de firewalls muito utilizada conhecida como proxy transparente. Optou-se pela localização dos proxies em nível de usuário por diversos motivos. O principal motivo é o fato de que o desenvolvimento de programas no espaço de kernel é muito complexo e limitado. Devido a essa complexidade e também pelo fato de ser muito difícil depurar programas no nível do kernel, muito bugs iriam surgir. O grande problema está no fato de que a presença de bugs no kernel da máquina representa um grave problema de segurança. Portanto, uma solução de segurança complexa como a proposta não deve ser implementada em nível de kernel. Sabe-se que a performance será penalizada devido ao aumento do número de trocas de contexto, porém esse é um preço que deve ser pago em

favor da segurança obtida pelo sistema. Outro fato que contribuiu para optarmos por uma solução em nível de usuário é o fato de já existirem, no mercado, diversas bibliotecas de código aberto com rotinas criptográficas e de comunicação em rede bastante desenvolvidas e testadas. Isso torna o processo de desenvolvimento mais rápido e diminui os riscos deste projeto, tendo em vista que o tempo disponível para a implementação é reduzido.

Para o usuário, esse sistema é quase que totalmente transparente conforme desejava-se pelo requisito 5. O usuário acessa os arquivos da mesma forma que em um servidor NFSv3 tradicional. O único impecilho é a necessidade do usuário digitar sua senha novamente para logar-se ao proxy cliente. Isso será melhor discutido mais adiante quando apresentarmos o mecanismo de autenticação. Porém, existe a possibilidade de eliminar esse impecilho se integrarmos nossa solução com o PAM (Pluggable Authentication Module), que é o módulo do Linux responsável pela autenticação dos usuários. É possível integrar o proxy cliente no PAM de forma que, ao logar-se na máquina, o usuário automaticamente estará se logando no proxy e já terá acesso aos seus arquivos remotos.

O proxy cliente intercepta todo o tráfego NFS, desde a primeira chamada ao portmapper no momento da montagem. O proxy do lado cliente então envia essa requisição para o proxy server do servidor destino para que este contacte o portmapper do servidor. Quando a resposta da requisição retorna ao proxy cliente ele descobre o número da porta na qual o programa Mountd do servidor está escutando. Sendo assim o proxy client insere uma nova regra no filtro local para capturar as próximas requisições que o cliente NFS faria ao mountd do servidor. Novamente o proxy intermedia essa comunicação levando a requisição até o servidor, e retornando o descritor de arquivos para que o cliente NFSv3 possa acessar a hierarquia montada. O proxy cliente manterá uma única conexão TCP com o servidor, e todas as requisições proveniente de todos os usuários serão tuneladas por essa conexão. Na verdade, todas as requisições são feitas pelo módulo do NFSv3 presente na camada VFS (Virtual File-System) do sistema operacional. Essas requisições são sempre realizadas tento uma porta de origem menor que 1024, que são as portas privilegiadas. Dessa maneira, o proxy poderá identificar quais são as requisições legítimas originadas pelo kernel, e as requisições montadas em nível de aplicação. Sendo assim, o proxy client só atenderá as requisições originadas de portas privilegiadas.

4.3 Mecanismos de segurança

Para atender os requisitos 6,7,8, e 9 que referem-se à segurança, faz-se necessário o uso de mecanismos criptográficos que possam garantir cada um dos itens referidos. Ao invés de propormos algoritmos e mecanismos novos de criptografia, procurou-se tentar encontrar dentre as soluções existentes e bem conhecidas, aquelas que se adequem aos nossos desejos. A ciência da criptografia obteve um grande avanço nos últimos 30 anos, apresentando hoje soluções bastantes seguras para os problemas levantados nesses requisitos. Para resolver o caso da autenticação do usuário (requisito 6), que é o problema mais crítico, foi feita a opção por utilizar o Kerberos V5. Essa também é a solução escolhida por outros sistemas de arquivos como o NFSv4, CIFS, Coda e o AFS. O Kerberos é amplamente difundido e está presente em diversas plataformas. Inclusive o Windows, a partir da versão 2000, utiliza Kerberos como forma padrão de autenticação. O Kerberos é capaz de utilizar segredos pequenos (como senhas) para realizar a autenticação do usuário diferentemente de certificados, que, em geral, trabalham com chaves de 512 bits. Além disso, o Kerberos é a única solução segura de autenticação que utiliza somente criptografia simétrica. Isso contribui muito para a performance do sistema, haja vista que a criptografia simétrica é duas ordens de grandeza mais rápida do que a criptografia assimétrica. O Kerberos também autentica o servidor, portanto já está sendo atendido o requisito 7. Porém o fator mais relevante da utilização do Kerberos é que, após obtido o TGT (Ticket-granting Ticket), o cliente pode obter um ticket para acessar o servidor sem a necessidade de utilizar o segredo novamente. Isso é ideal para o nosso cenário pois o cliente não deve ficar sendo interrompido para entrar com a sua senha a todo momento, e também não é aconselhável que a senha do cliente fique armazenada em memória para futuras utilizações. O cliente que quiser acessar seus arquivos no servidor nfs deve executar um programa que solicita a senha do usuário e contata o Kerberos para obter um TGT. A senha do usuário é deletada da memória, porém o TGT é armazenado no proxy cliente. A partir daí, toda requisição proveniente deste usuário que chegar ao proxy cliente fará uso desse TGT para obter os tickets para acessar qualquer servidor NFS da rede. Nenhum outro usuário da máquina, com exceção do super-usuário, terá acesso a esse TGT e portanto não poderá acessar os arquivos deste usuário. Quando o proxy recebe uma requisição, ele verifica se existe em seu cachê algum TGT pertencente ao usuário identificado na requisição do NFS, e caso não possua ele recusa a requisição. O Kerberos adiciona um obstáculo nessa solução que

é a configuração do servidor de autenticação, chamado KDC (Key Distribution Center). Porém esta é a única solução que apresenta um nível alto de segurança para o nosso cenário, portanto não há como se evitar.

A autenticação da máquina cliente (requisito 8) não é garantida pelo Kerberos. O Kerberos consegue autenticar o usuário e servidor NFS. A solução é utilizar certificados digitais. Porém, essa solução será oferecida de modo opcional pois a configuração desse recurso é mais trabalhosa e, conforme já foi mencionado, muitas redes podem não precisar autenticar as máquinas clientes. Pode se utilizar um certificado auto-assinado que ficará armazenado no servidor, e utilizando a chave privada correspondente, serão assinados os certificados de todas as máquinas clientes. Isso será terá de ser realizado manualmente pois qualquer tentativa de automatizar o processo abrirá brechas para que um invasor consiga assinar seus próprios certificados. Dessa maneira, de posse do certificado público da entidade certificadora (que é na verdade uma CA virtual), o servidor poderá verificar a validade dos certificados das máquinas clientes.

Quanto à privacidade e integridade dos dados. Isso pode ser feito através da própria chave de sessão estabelecida pelo Kerberos. O Kerberos atualmente suporta quatro algoritmos criptográficos: DES, 3-DES, RC4 e AES (Rindjael). O DES é um algoritmo considerado inseguro para o estado atual dos computadores. O 3-DES, apesar de ser computacionalmente custoso, é considerado bastante seguro e é uma opção viável de utilização. Já o RC4 é considerado um algoritmo pouco seguro e ainda enfrenta obstáculos relativos a patentes. O algoritmo mais recomendado é o AES devido à sua segurança e performance adequada.

Capítulo 5

Implementação

Para fins de prova de conceito foi implementado um protótipo da solução apresentada no capítulo anterior. O resultado foi um protótipo funcional de aproximadamente 2 mil linhas de código. A linguagem utilizada foi C pois a maior parte das tecnologias utilizadas (Kerberos, criptografia, interceptação de pacotes, sockets etc) já possuem bibliotecas escritas nessa linguagem. Além disso é uma linguagem que atende aos requisitos de performance que o nosso sistema necessita. A plataforma adotada foi o GNU/Linux (kernel 2.6) e a distribuição utilizada no desenvolvimento foi o Fedora Core versões 2 e 3.

A maior dificuldade encontrada nessa implementação foi a falta de documentação de algumas tecnologias utilizadas, em especial, a biblioteca do Kerberos 5 e a biblioteca LibIpq. A biblioteca LibIpq é uma biblioteca que permite interceptar pacotes diretamente no Netfilter (arcabouço de filtragem de pacotes do Linux) para tratá-los em nível de usuário. Essa falta de documentação foi o principal motivo que fez com que essa fase de implementação tomasse mais tempo do que o previsto. Essas duas bibliotecas possuem uma documentação muito escassa dificultando demais o processo de aprendizado da tecnologia. No caso da biblioteca do Kerberos 5, o problema foi mais grave por ser uma biblioteca muito extensa com mais de 200 funções. A única documentação existente dessa biblioteca é um guia de referência, que acompanha a biblioteca, descrevendo a API (interface de programação) de cada uma das funções. Porém o texto é muito sucinto e não tem nenhuma finalidade didática. Não foi encontrado nenhum livro, artigo ou documento que fornecesse informações de como utilizar essa biblioteca. O método de aprendizado utilizado foi através de engenharia reversa de programas prontos que já utilizam a biblioteca

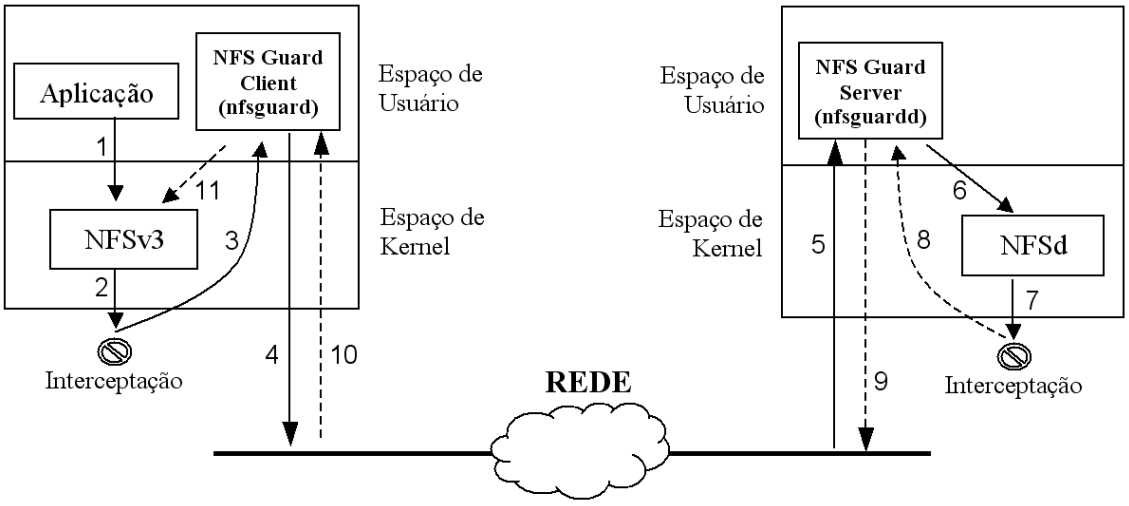

Figura 5.1: Arquitetura do NFS Guard

e também através do próprio código-fonte da biblioteca.

5.1 Arquitetura do NFS Guard

A figura 5.1 mostra a arquitetura do NFS Guard. Acompanhando as setas numeradas na figura, descreve-se o funcionamento dessa arquitetura.

1. A aplicação executa uma chamada read( ) ao sistema de arquivos do NFSv3.
2. O módulo do NFS do S.O. realiza uma chamada RPC_read() para o servidor NFS
3. Esse pacote UDP é interceptado pelo filtro e entregue para o nfsguard (proxy cliente) que se encontra em espaço de usuário
4. O nfsguard verifica em seu cache se o UID (número de usuário) contido na chamada RPC_read possui um ticket para acessar o servidor. Se o UID possuir um ticket, então a chamada RPC é cifrada com a chave de sessão associada a esse ticket e enviada para o servidor.
5. A mensagem é recebida pelo nfsguardd (proxy server).
6. A mensagem é decifrada e é verificada a sua integridade. Se tudo estiver certo, a chamada RPC_read() é repassada para o NFSd. Note que, para fazer esse repasse,

é preciso que o nfsguardd envie um pacote com o endereço de origem do cliente, e a única forma de forjar essa informação é através de raw sockets. Raw socket é um tipo de socket onde pode-se montar manualmente todos os cabeçalhos dos protocolos de rede, inclusive o IP onde se encontra o endereço de origem.

7. O Servidor NFS retorna a resposta à chamada RPC_read()

8. Esse pacote UDP é interceptado pelo filtro e entregue ao nfsguardd que se encontra em espaço de usuário

9. O nfsguardd cifra a resposta usando a mesma chave de sessão utilizada na seta 4

10. A mensagem é recebida pelo nfsguard

11. A mensagem é decifrada e sua integridade conferida. Somente então a resposta é repassada para o cliente. Nesse passo também são utilizados raw sockets.

Um fator que atrasou a implementação foi a escolha da tecnologia de interceptação dos pacotes. Inicialmente a idéia era utilizar NAT (Network Address Translation) que é a mesma tecnologia utilizada em proxies transparentes. Chegou-se a implementar um protótipo usando essa tecnologia, porém após vários testes mal sucedidos e investigação da causa chegou-se à conclusão que essa tecnologia não seria viável. O NAT faz a troca do endereço de destino quando o pacote passa pelo firewall, desviando o pacote para o proxy. O proxy transparente, por sua vez, precisa saber qual era o endereço de destino original daquele pacote modificado e para isso possui um comando que faz esse mapeamento. Quem faz esse mapeamento é o módulo ip_conn_track, que guarda os destinos originais dos pacotes que passaram pelo NAT. O problema é que o ip_conn_track só faz esse mapeamento para pacotes TCP. Para pacotes UDP que é o caso do NFS, esse recurso de proxy transparente não funciona.

Em seguida tentou-se erroneamente utilizar a biblioteca LibPCap para fazer a interceptação dos pacotes. A biblioteca LibPCap (Packet Capture Library) é uma biblioteca extremamente portável destinada à captura de pacotes. Programas consagrados como tcpdump, ethereal e snort utilizam essa biblioteca. Chegou-se até a fazer um protótipo de tunelamento completo usando essa tecnologia. Porém, somente mais tarde descobriu-se que esse mecanismo não seria adequado. A libpcap é uma biblioteca destinada para

fazer somente a captura dos pacotes mas precisava-se realmente interceptar o pacote, ou seja, não só capturar mas também impedir que o pacote seguisse o seu rumo. Tentou-se utilizar regras de firewall para tentar impedir o prosseguimento dos pacotes, porém a libpcap não consegue enxergar mais os pacotes quando os mesmos são filtrados. O ponto onde a libpcap captura os pacotes fica posterior à filtragem de pacotes. Então com esse mecanismo, ou se captura os pacotes deixando-os prosseguir seu rumo, ou não se captura nada. Portanto, esse método não nos seria útil para o que pretende-se.

A única solução encontrada para realizar essa interceptação dos pacotes foi através do módulo IP Queue do Netfilter. Este módulo é parte integrante do pacote Netfilter e seu objetivo é oferecer suporte para a realização de filtragem de pacotes em nível de usuário. Os pacotes são interceptados e repassados para um programa de nível de usuário que decide se o pacote deve ser aceito ou rejeitado. A biblioteca LibIpq é uma biblioteca que possui funções e estruturas prontas para utilização do módulo IP queue. Portanto, essa foi a tecnologia utilizada para interceptarmos as requisições nos passos 3 e 8 da figura 5.1. Ao passar pelo netfilter, os pacotes eram desviados pelo módulo ip queue para aplicação que, por sua vez, informava ao netfilter para que ele rejeitasse o pacote. Porém a aplicação tomava para si a responsabilidade de enviar esse pacote ao seu destino. Para utilização dessa biblioteca é necessário que o programa seja executado como super-usuário (root) pois o módulo ip queue restringe isso por medida de segurança.

5.1.1 Mecanismos de Segurança Implementados

O NFS Guard utiliza Kerberos (versão 5) como mecanismo de autenticação. A implementação do Kerberos utilizada foi a implementação oficial do MIT que é a implementação mais utilizada e difundida. Por enquanto a implementação utiliza o 3-DES como algoritmo de cifragem. Como o suporte ao AES do Kerberos foi implementado recentemente, seu código não foi amplamente testado e, portanto, pode apresentar falhas que atrapalhariam o desenvolvimento da nossa ferramenta. Para minimizar o risco, optou-se por utilizar o 3-DES provisoriamente e, assim que o código do NFS Guard estiver estabilizado, será passado a utilizar o AES. Para mudar do 3-DES para o AES, não será necessária nenhuma mudança no NFS Guard. Basta alterar as configurações do KDC para gerar tickets usando o AES. Para garantir integridade e privacidade utilizaram-se as funções de cifragem e assinatura digital que estão incluídas na própria biblioteca do Kerberos.

Através dessas funções a nossa ferramenta fica independente do algoritmo utilizado por trás dela. Portanto, o NFS Guard irá funcionar corretamente com qualquer algoritmo de cifragem e integridade suportado pela biblioteca do Kerberos, sem necessitar qualquer alteração ou configuração. A própria biblioteca já se encarrega de descobrir qual o algoritmo associado ao ticket e utilizá-lo automaticamente nas operações de cifragem e assinatura digital. Atualmente os algoritmos suportados são:

- Cifragem: DES, 3-DES, RC4, AES-128, AES-25
- Integridade: CRC-32, MD4, MD5, SHA1

O NFS Guard faz autenticação mútua das partes através do Kerberos. Isso significa que o usuário se autentica perante o servidor e vice-versa. Portanto, o cliente também certifica-se de que está falando com o servidor correto. Isso evita que um atacante possa personificar um servidor.

Não foi implementado o mecanismo de certificados proposto para o requisito 8. Por ser um requisito não obrigatório, optou-se por atribuir menor prioridade a essa etapa da implementação e não foi possível concluí-la.

A implementação do NFS Guard garante autenticidade, integridade e privacidade de forma obrigatória. Na forma que foi implementado não é possível

É possível alterar a implementação para que se possar configurar o NFS Guard para prover integridade com privacidade, mas também é possível optar somente pela integridade. A integridade e a autenticação continuam sendo providas, mas os dados são transmitidos em texto claro pela rede. Essa opção é menos segura, porém a performance é sensivelmente superior pois os dados não são mais encriptados.

5.2 Visão detalhada da Implementação

5.2.1 Implementação do lado cliente (nfsguard)

O nfsguard utiliza uma arquitetura com múltiplos processos onde cada processo filho é responsável pela comunicação com um determinado servidor. Essa abordagem facilita a implementação e contribui para estabilidade do sistema. Se houver falha em um determinado servidor, as demais comunicações não serão afetadas. Caso fosse feita a opção

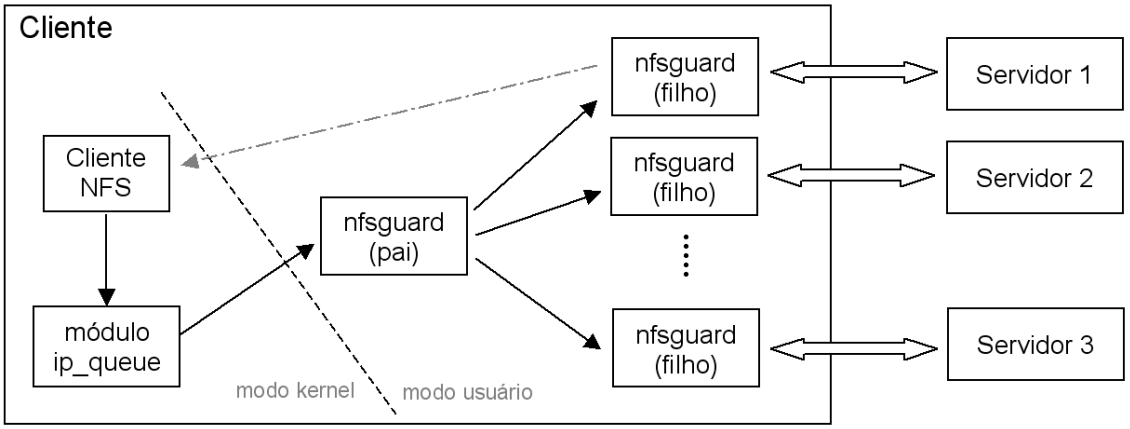

Figura 5.2: Arquitetura do cliente NFS Guard

por uma arquitetura monolítica seria necessário utilizar sockets não-bloqueantes para que o nfsguard não ficasse bloqueado tentando enviar um pacote para um servidor fora de operação. Porém a utilização de sockets não-blockeantes é trabalhosa e foi evitada. Na arquitetura proposta, cada processo lida apenas com uma conexão TCP e o processo é encerrado caso esta conexão seja encerrada. Uma possibilidade de implementação seria a utilização de processos leves (*threads*), que proporcionariam uma melhor performance ao sistema. Entretanto, por se tratar apenas de um protótipo, optou-se por utilizar processos independentes devido à facilidade de implementação e depuração.

A figura 5.2 mostra a arquitetura interna do cliente NFS Guard. A requisição RPC originada do cliente NFS é interceptada em nível de rede e processada pelo módulo ip_queue do Linux. O pacote é então capturado e levado para o nível de usuário pelo nfsguard pai através da biblioteca libipq. O processo nfsguard pai então verifica, através do cabeçalho do pacote capturado, qual é o servidor destino desta requisição e, em seguida, repassa o pacote para o nfsguard filho responsável. Caso este pacote seja o primeiro contato com um determinado servidor, não haverá processo filho instanciado, logo o processo pai irá realizar uma chamada de sistema fork() para criar este novo processo.

Através de um *pipe* o nfsguard filho recebe o pacote do processo pai e estabelece uma conexão TCP com o servidor nfsguardd. Em seguida, o processo filho verifica se o usuário identificado na requisição RPC (campo UID) possui um ticket Kerberos no sistema.

Os tickets Kerberos são obtidos através do comando kinit na linha de comando, ou através do módulo Kerberos para o PAM. De qualquer uma dessas formas, o ticket TGT

é obtido junto ao KDC e é armazenado no diretório /tmp do sistema com permissões de leitura somente para o referido usuário.

O nfsguard filho verifica se o arquivo contendo o ticket se encontra no sistema de arquivos. O processo então contacta o servidor TGS, e usando o TGT do usuário obtém o ticket de fato para acessar o servidor NFS Guard. Em seguida, o ticket é utilizado para autenticar-se perante o servidor NFS Guard, e a chave de sessão contida no ticket é utilizada para cifrar todo o tráfego.

Todos os tickets são armazenados em memória (mecanismo de *cache*) para eliminar os impactos de acesso ao disco e ao servidor de autenticação. Desta forma, a primeira requisição sofrerá os impactos de acesso aos recursos, porém o tráfego maciço subseqüente não será penalizado. Esse mecanismo é implementado através de uma tabela que relaciona UID e ticket correspondente.

O processo de comunicação inverso se dá da seguinte maneira. As respostas vindas do servidor são recebidas pelo processo filho e são decifradas. Dessa forma obtém-se o pacote IP original enviado pelo servidor NFS. Esse pacote é enviado diretamente para o cliente NFS usando *raw sockets*, que é uma forma de utilização de sockets que permite a montagem completa de pacotes IP em nível de usuário. Portanto, o processo nfsguard filho estará fazendo uso de uma técnica conhecida como *IP spoofing*, se fazendo passar pelo servidor, enviando pacotes com endereço de origem do mesmo. Para a utilização dessa técnica é necessário que o processo possua privilégios de super-usuário.

A comunicação entre o processo pai e o processo filho se dá através do mecanismo de *pipes*, padrão em sistemas Unix. O processo pai mantém uma tabela atualizada constantemente que relaciona o IP do servidor destino com o número do processo (PID) filho responsável.

O impacto de criação e encerramento de processos no sistema é muito pequeno, pois o processo filho só é criado no momento do primeiro contato com um determinado servidor. Nas requisições subseqüentes não será necessário instanciar um novo processo.

O processo filho possui um temporizador de inatividade que, após um tempo configurável (padrão é 10 minutos), automaticamente encerra a conexão com o servidor e se auto-encerra. O processo pai é notificado através de sinais, e exclui a respectiva entrada de sua tabela. Portanto, caso um cliente fique inativo por um certo tempo, o primeiro pacote sofrerá um atraso devido à criação do processo, porém todo o tráfego subseqüente

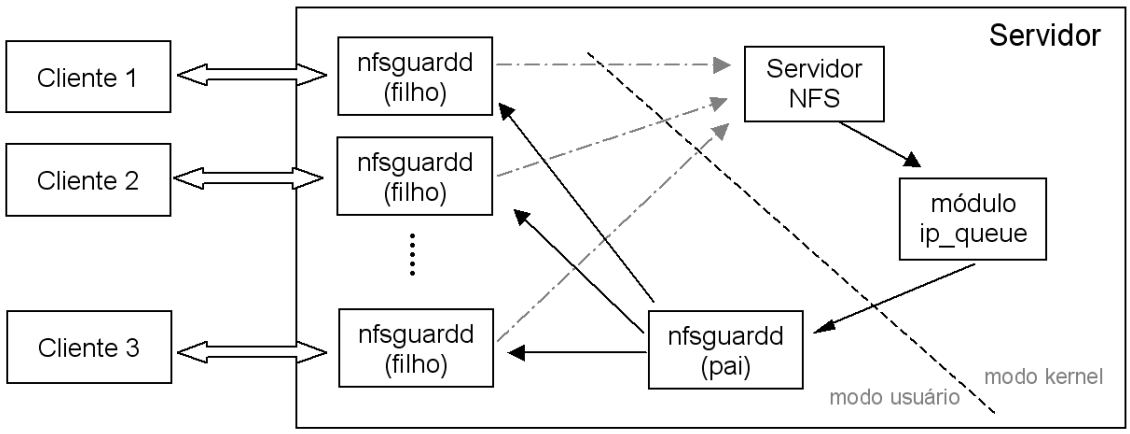

Figura 5.3: Arquitetura do servidor NFS Guard

não sofrerá esse impacto. Esse sistema de temporização foi implementado para poupar recursos que não estejam sendo utilizados.

A arquitetura do NFS Guard proporciona um funcionamento totalmente transparente. O cliente NFS envia suas requisições e recebe as respostas normalmente como em uma rede NFS padrão, sem ter ciência da presença do NFS Guard.

5.2.2 Implementação do lado servidor (nfsguardd)

O servidor NFS Guard apresenta diversas similaridades ao cliente visto anteriormente. A implementação também faz uso de uma arquitetura *multithread* para tratar as requisições. A figura 5.3 mostra a arquitetura interna do servidor NFS Guard.

O servidor nfsguardd é responsável por receber as novas conexões TCP. A porta TCP escolhida foi a porta 40 porém isso pode ser facilmente alterado. Ao receber uma nova conexão o processo pai instancia um processo filho através da chamada de sistema fork().

O processo filho recebe os dados diretamente do cliente. O nfguardd filho é responsável por checar as credenciais do cliente. Para isso, basta verificar se o cliente possui um ticket válido. Se a autenticação for bem sucedida, o processo irá decifrar a requisição e enviá-la diretamente ao servidor NFS usando *raw sockets*.

Além de receber novas conexões, o processo pai também é responsável por interceptar as respostas do servidor NFS usando libipq e, de acordo com o IP destino contido no pacote, repassar para os processos filhos responsáveis.

Os processos filhos recebem a resposta do servidor e, cifrando com as chaves ade-

quadas, enviam de volta para o cliente correspondente. Nesta etapa há um problema a ser contornado. No cabeçalho RPC há a identificação do usuário (UID) somente nas requisições, mas nas respostas não há essa informação. Dessa forma, seria impossível saber a qual usuário pertence uma dada resposta, logo não haveria como saber qual chave de sessão utilizar. Para resolver este problema é preciso que o servidor mantenha o estado das requisições para ser capaz de identificar os usuários destinatários das respostas. Para isso, foi implementado um mecanismo que gerencia uma tabela cuja função é relacionar o UID e o identificador de transação (XID) que é um número único para cada requisição. Ao processar uma requisição, seu UID e XID são inseridos na tabela. No momento processar uma resposta, pode-se descobrir seu UID a consultando a tabela com o XID respectivo.

O servidor NFS guard também atua de maneira transparente. O servidor NFS não tem conhecimento da presença de todo o sistema intermediário envolvido. Logo, não é necessária nenhuma alteração administrativa no serviço NFS para que ele funcione com o NFS Guard.

5.3 Instalação e Configuração do NFS Guard

Os requisitos de software são bastante modestos. Basicamente a única coisa que deve ser instalada nas máquinas é a biblioteca do Kerberos. A maioria das distribuições Linux já traz esse pacote.

O NFS do linux apresenta um problema de implementação que foi descoberto nos testes. Mesmo quando, na operação de montagem (comando mount), é especificada a utilização apenas do protocolo UDP, o cliente NFS tenta procurar por um portmapper TCP. Para resolver esse inconveniente, providenciou-se um work-around simples que consiste em um falso portmapper que executa localmente no cliente. Quando o cliente tenta acessar o portmapper de algum servidor, essa requisição é desviada para um portmapper falso. Esse programa foi denominado fakeportmapper e ele sempre dá uma resposta nula para qualquer pergunta dizendo que não há serviços registrados. Desse modo consegue-se "enganar"o cliente NFS do Linux e forçar para que ele utilizasse somente o UDP.

Ainda não foram desenvolvidos scripts de instalação e de inicialização. Dessa forma, essas tarefas devem ser feitas manualmente, o que não representa muito inconveniente pois as tarefas são simples e é trivial colocá-las em um script.

As instruções para executar o lado cliente do NFS Guard são as seguintes.

1. Carregar o modulo ip_queue

```
% modprobe ip_queue
```

2. Inserir as regras para interceptar os pacotes

```
% iptables -A OUTPUT -o eth0 -p udp --sport 0:1023 --dport $PORTMAPPORT -j QUEUE;
% iptables -A OUTPUT -o eth0 -p udp --sport 0:1023 --dport $MOUNTPORT -j QUEUE;
% iptables -A OUTPUT -o eth0 -p udp --sport 0:1023 --dport $NFSPORT -j QUEUE;
```

3. Inserir a regra para o Fake Portmapper

```
% iptables -t nat -A OUTPUT -p tcp --dport 111 -j DNAT --to-destination 127.0.0.1:112;
```

4. Rodar o Fake portmapper

```
% fakeportmap &
```

5. Rodar o NFS Guard (proxy cliente)

```
% nfsguard &
```

As instruções para executar o lado servidor do NFS Guard são as seguinte:

1. Carregar o modulo ip_queue

```
% modprobe ip_queue
```

2. Inserir as regras para interceptar os pacotes

```
% iptables -A OUTPUT -o eth0 -p udp --sport 0:1023 --dport $PORTMAPPORT -j QUEUE;
% iptables -A OUTPUT -o eth0 -p udp --sport 0:1023 --dport $MOUNTPORT -j QUEUE;
% iptables -A OUTPUT -o eth0 -p udp --sport 0:1023 --dport $NFSPORT -j QUEUE;
```

3. Bloquear o acesso ao NFS sem passar pelo NFS Guard

```
% iptables -A INPUT -i eth0 -p tcp --dport $PORTMAPPORT -j DROP;
% iptables -A INPUT -i eth0 -p tcp --dport $MOUNTPORT -j DROP;
% iptables -A INPUT -i eth0 -p tcp --dport $NFSPORT -j DROP;
% iptables -A INPUT -i eth0 -p udp --dport $PORTMAPPORT -j DROP;
% iptables -A INPUT -i eth0 -p udp --dport $MOUNTPORT -j DROP;
% iptables -A INPUT -i eth0 -p udp --dport $NFSPORT -j DROP;
```

4. Rodar o NFSGuardd (proxy server)

```
% nfsguardd &
```

Com apenas essas instruções todo o sistema do NFS Guard estará funcionando e todo o tráfego será desviado para dentro deste túnel cifrado. Isso inclui também as requisições ao portmapper e ao mountd. Para montar um diretório, basta entrar com o seguinte comando no cliente:

```
% mount -t nfs -o nolock,udp  <maquina>:<diretório> <ponto_de_montagem>
```

A opção udp determina que o comando mount deve montar a hierarquia utilizando o UDP como protocolo de transporte. A opção nolock desliga o protocolo NLM (Network Lock Manager) pois esse protocolo ainda não é suportado pelo NFS Guard.

A utilização por parte do usuário é transparente. A única ação diferente que o usuário deve fazer é autenticar-se manualmente perante o Kerberos, ou seja, obter um TGT. Isso é feito através do comando kinit. Basta o usuário digitar sua senha e o TGT é obtido automaticamente. É possível eliminar essa necessidade de autenticar manualmente através do módulo de kerberos para o PAM (Pluggable Authentication Modules) do Linux. Dessa forma o TGT já é obtido assim que o usuário loga no sistema.

5.4 Testes e Performance

Não houve tempo para testar e depurar suficientemente a implementação, portanto a estabilidade da implementação ainda está bastante problemática. Mesmo assim foi possível realizar alguns experimentos para analisar a performace geral do sistema. Havia uma grande expectativa em conhecer, ou ao menos se ter uma idéia, de qual seria a taxa de transferência máxima alcançada pelo NFS Guard.

O experimento foi realizado entre duas máquinas de configurações similares (AthlonXP 1800+ e Pentium 4 2.0GHz) interligadas por uma rede 100Mbps com switchs. Cada máquina possuia 512MBytes de mémoria e um disco IDE de 7200rpm. Ambas as máquinas utilizavam Linux Fedora Core 2 com kernel 2.6.

Visando isolar ao máximo as variáveis, optou-se por não utilizar o disco no experimento. Ao invés disso, utilizou-se arquivos em memória através da partição /dev/shm

no Linux. Trata-se de uma partição mapeada em memória RAM que utiliza automaticamente os bytes livres da memória. O servidor NFS foi configurado para compartilhar essa partição onde foi colocado um arquivo de 100MBytes. A máquina cliente iria então realizar uma transferência desse arquivo para a sua partição /dev/shm. Portanto, em nenhum momento haveria acessos ao disco diretamente relacionados à essa transferência. Para medirmos o tempo de tranferência foi utilizado o programa time que é um utilitário tradicional no Unix que mede o tempo de execução de um processo. Ele foi executado para aferir o tempo tomado pelo processo cp para realizar a cópia do arquivo remoto para a partição local.

O experimento foi realizado para o NFS sobre UDP, NFS sobre TCP e para o NFS Guard. Foram realizadas várias aferições para cada sistema, chegando-se assim à média das aferições. Foram excluídas da média algumas medidas consideradas *outliers* (pontos fora da curva) pois essas medições apresentavam disparidades atribuídas a outras variáveis (como tarefas de outros processos, rotinas periódicas do kernel etc).

Os resultados são apresentado na tabela 5.1.

Sistema de Arquivos	Tempo Aferido (seg)	Taxa de Transferência (MB/seg)
NFSv3 (UDP)	9.76	10.24
NFSv3 (TCP)	10.32	9.69
NFS Guard	29.49	3.39

Tabela 5.1: Teste de performance do NFS Guard

O resultado do experimento mostra que o NFS Guard apresenta uma performace inferior ao NFS com UDP em uma razão de aproximadamente 3 vezes. Enquanto o NFSv3 sobre UDP transferiu o arquivo em uma média de 9.76 segundo, o NFS Guard levou 29.49 segundos.

Pode-se levantar três fatores que impactam na performance do NFS Guard em relação ao NFS. O primeiro deles é o overhead imposto pela criptografia. Apesar de somente utilizar criptografia simétrica as operações criptográficas são computacionalmente custosas. O segundo fator é o fato do NFS Guard estar em nível de usuário, o que exige o dobro de trocas de contexto entre kernel e aplicação. O terceiro fator é que a implementação do NFS Guard realizada não possui qualquer otimização voltada para a performance.

Em qualquer sistema computacional, para se obter ganhos de segurança paga-se o preço da diminuição de performance. Logo é perfeitamente aceitável que um sistema seguro ofereça uma performance reduzida. Porém, ainda pode-se realizar melhorias no protocolo e principalmente na implementação que poderiam diminuir bastante essa disparidade.

Capítulo 6

Conclusão

Ao final deste trabalho chegamos a uma solução que atende aos requisitos e consegue elevar o nível de segurança para um nível bastante satisfatório. Desse modo, considera-se que os objetivos deste trabalho foram alcançados.

Um protótipo foi implementado com sucesso para fins de prova de conceito, porém foi pouco testado e tem sua estabilidade comprometida. Para que essa implementação possa ser realmente utilizada em ambientes de produção serão necessários mais alguns ajustes e melhorias, além de elaboração da documentação do programa.

A técnica de Kerberização através de interceptação de pacote redirecionando-os para o nível de usuário se mostrou bastante viável. Sendo possível utilizá-la em outros protocolos além do NFS. A grande vantagem dessa técnica é o fato de não requerer nenhuma modificação nas aplicações cliente e servidoras originais atuando de forma quase transparente. Logo é uma técnica que se integra facilmente a uma rede já instalada pois não há a necessidade de qualquer alteração administrativa nos serviços. Além disso essa técnica permite a utilização de nossa solução com software de código fechado. Como não foi encontrado nenhum registro de outra ferramenta que utilizasse tal técnica, considero que essa técnica atribui características inovadoras ao trabalho.

Temia-se que o impacto na performance pudesse ser grande demais a ponto de inviabilizar a utilização do NFS Guard, porém a performance da ferramenta se mostrou bastante adequada, principalmente quando se leva em conta os benefícios que a ferramenta traz. Não houve tempo para realizarmos testes rigorosos de performance, mas os experimentos mostraram um impacto significativo na utilização do NFS Guard (queda de aproxima-

damente um terço desempenho) em relação ao NFS nativo. Essa medida foi realizada usando o 3-DES como algoritmo de criptografia, porém, como já foi dito, esse algoritmo tem uma performance bastante lenta. Dessa forma, espera-se que a perfomance possa ser melhorada com a utilização do algoritmo AES.

O NFS Guard é uma solução viável para agregar segurança a uma rede que utiliza o NFSv3. Porém em sistemas onde o uso do NFS não é um requisito, é aconselhável utilizar o AFS ou CIFS (com Kerberos), pois são sistemas já conhecidos e consagrados que apresentam boa segurança e robustez. O NFSv4 também apresenta ótimas características de segurança porém é um sistema recente e que ainda não foi largamente testado. Suas implementações ainda são incompletas e instáveis, logo ainda é cedo para utilizá-lo em ambientes de produção. O sistema SHFS é uma solução simples, rápida e segura para acesso a sistema de arquivos remotos, podendo eliminar a necessidade de estabelecer uma VPN. Apesar se ser também um programa muito recente, o SHFS vem ganhando muito espaço e também merece bastante atenção.

6.1 Trabalhos futuros

Existem várias possibilidades de extensões ao projeto proposto nesta dissertação. A primeira delas seria o aprimoramento da implementação realizada. O programa ainda apresenta diversas falhas de programação que precisam ser depuradas para que o programa possa ser utilizado em ambientes de produção. Também seria necessário o desenvolvimento de utilitários de instalação e configuração do serviço, bem como a elaboração da documentação do programa.

Uma outra possibilidade de trabalho futuro seria a otimização da implementação para performance e o estudo da viabilidade de implementação em kernel. A implementação de serviços de criptografia no kernel é dificultada pelo fato de não ser possível utilizar as várias bibliotecas de funções criptográficas existentes em nível de usuário (como a libkrb5 utilizada neste trabalho). Contudo a execução de parte do NFS Guard em nível de kernel poderia trazer um ganho considerável de performance.

Um ponto que não foi abordado neste trabalho foi a integração do NFS Guard ao autofs e ao NLM (NFS Lock Manager). Seria necessário fazer um estudo da viabilidade de operação do NFS Guard em conjunto com esses serviços.

Outra tarefa a ser realizada futuramente seria a portabilidade deste código para outras plataformas. Conforme especificado nos requisitos, o NFS Guard foi projetado de forma que pudesse ser implementado em outros sistemas operacionais, e isso é necessário para que o sistema seja utilizado em redes heterogêneas.

Apêndice A

Glossário

3-DES	Triple DES
ACE	Access Control Entry
ACL	Access Control List
AES	Advanced Encryption Standard
AFS	Andrew File System
API	Application Programming Interface
CBC	Cipher Block Chaining
CIFS	Common Internet File System
CMU	Carnegie Mellon University
CRC	Cyclic Redundancy Check
DCE	Distributed Computing Environment
DEC	Digital Equipment Corporation
DES	Data Encryption Standard
DFS	Distributed File System
DH	Diffie-Hellman
DHCP	Dynamic Host Configutarion Protocol
DNS	Domain Name Service

FTP	File Transfer Protocol
GID	Group Identifier
GNU	GNU is Not Unix
GSS-API	Generic Security Services API
HMAC	Hashed Message Authentication Code
HTTP	Hypertext Transfer Protocol
IETF	Internet Engeneering Task Force
IP	Internet Protocol
KDC	Key Distribution Center
KML	Kernel Modification Log
LDAP	Lightweigth Directory Access Protocol
LGPL	Lesser General Public License
LIPKEY	Low Infrastructure Public Key Mechanism
MAC	Message Authentication Code
MD5	Message Digest 5
MIT	Massachusetts Institute of Technology
NAT	Network Address Translation
NFS	Network File System
NFSv2	Network File System Version 2
NFSv3	Network File System Version 3
NFSv4	Network File System Version 4
NLM	Network Lock Manager
NTLM	indows NT Lan Manager
PAM	Pluggable Authentication Modules
POSIX	Portable Operating System Interface
RC4	Rivest Cipher 4

RFC	Request for Calls
RPC	Remote Procedure Call
SCP	Secure Copy Protocol
SHFS	Secure Shell File System
SMB	Server Message Block
SNFS	Secure Network File System
SSH	Secure Shell
SSL	Secure Socket Layer
TCP	Transmition Control Protocol
TCSEC	Trusted Computer Security Evaluation Criteria
TGS	Ticket Granting Service
TGT	Ticket Granting Ticket
TNFS	Trusted Network File System
UDP	User Datagram Protocol
UID	User Identifier
VFS	Virtual File System
VPN	Virtual Private Network
VSG	Volume Storage Group
WWW	World Wide Web
XDR	External Data Representation

Referências Bibliográficas

[Agg87a] G.; et al. Aggarwal, A.; Arnold. *RFC1001 - Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts*. The Internet Engineering Task Force, 1987.

[Agg87b] G.; et al. Aggarwal, A.; Arnold. *RFC1002 - Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications*. The Internet Engineering Task Force, 1987.

[Bur04] Jesse Burns. Ntlm authentication unsafe. *Symposium on Security for Asia Network*, 2004.

[Cal95] B.; Staubach P.; Sun Microsystems Inc Callaghan, B.; Pawlowski. *RFC1813: NFS Version 3 Protocol Specification*. The Internet Engineering Task Force, 1995.

[Cal96a] B. Callaghan. *RFC2054: WebNFS Client Specification*. The Internet Engineering Task Force, RFC 2054, 1996.

[Cal96b] B. Callaghan. *RFC2055: WebNFS Server Specification*. The Internet Engineering Task Force, RFC 2055, 1996.

[CEC00] A. Chiu, M. Eisler, and B. Callaghan. *RFC2755: Security Negotiation for WebNFS*. The Internet Engineering Task Force, RFC 2755, 2000.

[Eis97] A.;Ling L. Eisler, M.;Chiu. *RFC2203: RPCSEC_GSS Protocol Specification*. The Internet Engineering Task Force, 1997.

[Eis00] M.; Zambeel; Eisler. *RFC2847: LIPKEY - A Low Infrastructure Public Key Mechanism Using SPKM*. The Internet Engineering Task Force, 2000.

[Fou93] Open Software Foundation. *OSF DCE Administration Reference*. Prentice Hall, 1993.

[Gar96] G. Garfinkel, S.; Spafford. *Practical Unix and Internet Security - Third Edition*. O'Reilly & Associates, 1996.

[Hag02] Bill von Hagen. Using the intermezzo distributed filesystem. *http://www.linuxplanet.com/linuxplanet/reports/4368/1/*, page Visitada em nov/05, 2002.

[Koh93] C. Kohl, J.; Neuman. *RFC1510: The Kerberos Network Authentication Service (V5)*. The Internet Engineering Task Force, 1993.

[Lim00] Paulo Lício Lima, Marcelo; Geus. Filtragem com estados de serviços baseados em rpc no kernel do linux. *II Simpósio Segurança em Informática*, 2000.

[Lin00] J.; RSA Labs; Linn. *RFC2078: Generic Security Services Application Program Interface*. The Internet Engineering Task Force, 2000.

[Mic88] Sun Microsystems. *RFC1050: RPC - Remote Procedure Call Protocol Specification*. The Internet Engineering Task Force, RFC 2054, 1988.

[Mor86] M. et.al. Morris, J. ; Satyanarayanan. Andrew: A distributed personal computing environment. *Commun. ACM*, 20:184–201, 1986.

[Nak03] Paulo Lício; Nakamura, Emílio; de Geus. *Segurança em Ambientes Corporativos*. Ed. Futura, 2003.

[O'S00] Declan O'Shanahan. Cryptosfs: Fast cryptographic secure nfs. Master's thesis, University of Dublin, 2000.

[Ros92] D.; Fisher G. Rosenberry, W.; Kenney. *Understandig DCE*. Digital Equipment Corporation, 1992.

[Sat90] J.; et. al Satyanarayanan, M.;Kistler. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39:47–59, 1990.

[SB91] M. Merrit S. Bellovin. Limitations of the kerberos authentication system. *USENIX Conference Procedings*, 1991.

[She00] B.; et al. Sheple, S.; Callaghan. *RFC3010: NFS version 4 Protocol*. The Internet Engineering Task Force, 2000.

[She03] B.; et al. Sheple, S.; Callaghan. *RFC3530: Network File System (NFS) version 4 Protocol*. The Internet Engineering Task Force, 2003.

[SM89] Inc. Sun Microsystems. *RFC1094: NFS - Network File System Protocol Specification*. The Internet Engineering Task Force, 1989.

[Ste88] C.;Schiller J Steiner, J.;Neuman. Kerberos: an authentication service for open network system. *Winter Usenix Conference*, 1988.

[Ste01] Mike; Labiaga Ricardo; Stern, Hal; Eisler. *Managing NFS and NIS - Second Edition*. O'Reilly & Associates, 2001.

[Woo87] J. P. L. Woodward. Security requirements for system high and compartemented mode workstations. Technical report, The MITRE Corporation, Belford MA, November, 1987.