Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise do OpenAFS sob a Perspectiva da Segurança em Sistemas de Arquivos Distribuídos

Fernando Marques Borges

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Eduardo Adilio Pelinson Alchieri

Brasília
2019

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

MBB732a
a

Marques Borges, Fernando
Análise do OpenAFS sob a Perspectiva da Segurança em Sistemas de Arquivos Distribuídos / Fernando Marques Borges; orientador Eduardo Adilio Pelinson Alchieri. -- Brasília, 2019.
104 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2019.

1. Sistemas de Arquivos Distribuídos. 2. Segurança da Informação. 3. Criptografia. I. Alchieri, Eduardo Adilio Pelinson, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise do OpenAFS sob a Perspectiva da Segurança em Sistemas de Arquivos Distribuídos

Fernando Marques Borges

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Eduardo Adilio Pelinson Alchieri (Orientador)
CIC/UnB

Prof. Dr. Marcos Fagundes Caetano
Universidade de Brasília

Prof. Dr. Edson Tavares de Camargo
Universidade Tecnológica Federal do Paraná

Prof.ª Dra. Aletéia Patrícia Favacho de Araújo
Coordenadora do Programa de Pós-graduação em Computação Aplicada

Brasília, 25 de janeiro de 2019

Dedicatória

Dedico esta dissertação com todo meu respeito e admiração aos meus pais Benício Pedro Borges e Maria Helena Marques Borges, que sempre me apoiaram e incentivaram meu crescimento pessoal e profissional.

Agradecimentos

Concluir esta dissertação foi tarefa árdua que não teria sido finalizada sem o apoio de várias pessoas, as quais passo a citar com profundo e sincero agradecimento.

Antes de mais nada, especial gratidão a Deus, que me concedeu o dom da vida. À minha familia e a todos os queridos amigos, sou muito grato pela sempre motivante torcida por meus projetos pessoais e profissionais.

Ao meu orientador, Prof Dr Eduardo Adilio Pelinson Alchieri, meu reconhecimento e sincero agradecimento pela paciência e, especialmente, pela compreensão com minhas dificuldades no decorrer da fase final de confecção do trabalho, nas diversas reuniões presenciais, bem como pela orientação segura em dezenas de ligações via Skype entre a África e o Brasil. Tudo me permitiu desenvolver e concluir este trabalho e, sem seu apoio e diretrizes, esta dissertação certamente não teria sido finalizada. Muito obrigado, Professor!

Ao Prof Dr Marcelo Ladeira e à Prof.ª Dra. Aletéia Patrícia Favacho de Araújo, agradeço pela compreensão e atenção dispensadas em todas as minhas solicitações junto ao PPCA.

Aos colegas de curso, agradeço pelo companheirismo e compartilhamento de informações. O espírito de corpo do grupo foi um importante motivador neste caminho.

Finalmente, agradeço aos senhores Coronéis Fernando Costa Adam e Jacy Barbosa Junior, meus ex-comandantes, que permitiram e incentivaram meu ingresso no Mestrado, em nome dos quais agradeço ao Exército Brasileiro pela oportunidade de participar do PPCA. Ainda, ao senhor Coronel Moises da Paixão Junior, meu atual chefe e grande incentivador, meu reconhecimento por todo apoio e incentivo recebidos no decorrer da confecção desta dissertação.

Resumo

As mais recentes abordagens ligadas à segurança em Sistemas de Arquivos Distribuídos (SAD) envolvem a aplicação de técnicas que garantam rigorosos níveis de disponibilidade, integridade, confidencialidade e autenticidade de dados em sistemas de armazenamento, estejam estes dados em trânsito ou em repouso. O papel da segurança é vital para o controle de uma variedade de desafios como intrusões, comportamentos maliciosos e monitoramento não-autorizado de tráfego. Este trabalho analisa os Sistemas de Arquivos Distribuídos sob a ótica da segurança, apresentando uma comparação entre os principais SAD. Apresenta, então, a análise de segurança do OpenAFS, com o propósito de considerá-lo como alternativa viável para implementação segura, alinhada às melhores práticas de segurança, escalabilidade e transparência.

Palavras-chave: Sistemas de Arquivos Distribuídos, Segurança da Informação, Criptografia

Abstract

The latest approaches related to security in Distributed File Systems (DFS) involves application of techniques that ensure strict levels of availability, integrity, confidentiality and authenticity of data in storage systems, whether in transit or at rest. The role of security is vital to control a variety of challenges such as intrusions, malicious behavior and unauthorized traffic monitoring. This work analyzes the Distributed File Systems from the perspective of security, presenting a comparison between the main DFS. Therefore, presents a security analysis of OpenAFS, with the purpose of considering it as an viable alternative for secure implementation, with best security practices, scalability and transparency.

Keywords: Distributed File Systems, Information Security, Cryptography

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACL Access Control Lists.

AFS Andrew Distributed File System.

API Application Programming Interface.

AS Authentication Server.

CBC Cipher Block Chaining.

CRC Cyclic Redundancy Check.

DES Data Encryption Standard.

DoS Denial of Service.

GFS Google File System.

HDFS Hadoop Distributed File System.

IBM International Business Machines.

KDC Key Distribution Center.

NAS Network-Attached Storage.

NFS Network File System.

RFC Request for Comments.

RPC Remote Procedure Call.

RX Extended Remote Procedure Call.

SAD Sistema de Arquivos Distribuídos.

SD Sistemas Distribuídos.

SI Segurança da Informação.

SSH Secure Shell.

TGS Ticket Granting Server.

TLS Transport Layer Security.

UDP User Datagram Protocol.

VLDB Volume Location Database.

VM Virtual Machine.

VSF Virtual File System.

Capítulo 1

Introdução

Sistemas Distribuídos (SD) exercem importante papel na computação moderna e são definidos, de modo geral, como múltiplos equipamentos independentes que, interligados, coordenam-se por meio de mensagens e comportam-se frente ao cliente como um único elemento. Raynal [6] define formalmente os SD como "um conjunto finito de processos independentes que se comunicam usando troca de mensagens, colaborando para a realização de alguma tarefa".

Várias são as motivações para utilização dos Sistemas Distribuídos, podendo ser citadas, dentre outras: facilitar compartilhamento de recursos (dados e dispositivos), prover transparência ao acesso e possibilitar a integração entre recursos heterogêneos.

O conceito de transparência é muito importante para o entendimento dos SD. Dada esta relevância, cabe citar Coulouris et al. [7], que assim o definem: "Transparência é a ocultação do usuário e do programador da aplicação da separação de componentes em um sistema distribuído, de modo que o sistema seja percebido como um todo, e não como uma coleção de componentes independentes".

Em um primeiro olhar, as motivações para o uso de SD são basicamente as de simplificar e facilitar o acesso aos sistemas. No entanto, em uma segunda dimensão, é inevitável constatar que há implicações relacionadas à segurança dos dados nestes sistemas exatamente pela intrínseca facilidade de acesso fornecida.

A propósito da segurança, discorreram ainda Coulouris et al. [7]: "Há uma necessidade generalizada de adoção de medidas para garantir a privacidade, integridade e disponibilidade dos recursos em SD". Os autores seguem enfatizando, neste contexto, que a proteção dos dados em Sistemas Distribuídos depende da adoção de políticas e mecanismos de segurança.

Tais fatores serão tratados na Seção 2.5 deste trabalho, mas cabe introduzir, neste ponto, que a implementação de mecanismos de segurança em Sistemas Distribuídos re-

quer gerência precisa e eficiente dos recursos, que pode ser definida por meio de análises rigorosas a respeito de potenciais ataques ou ameaças em variados contextos.

Casos particulares de Sistemas Distribuídos, criados para exercer funções específicas de compartilhamento de arquivos em rede, são denominados Sistema de Arquivos Distribuídos (SAD). O principal objetivo destes sistemas é o de prover acesso, de forma transparente, a arquivos remotamente distribuídos.

Ainda sob a perspectiva de Coulouris et al. [7], os SAD são sistemas que suportam o armazenamento persistente dos dados e programas de todos os tipos e a distribuição consistente de dados atualizados, provendo acesso a arquivos armazenados remotamente com desempenho e confiabilidade semelhantes a dados armazenados localmente. Em outras palavras, são sistemas criados para compartilhar dados de tal sorte que a localização destes não seja uma preocupação para o cliente que os acessa.

Nesta mesma linha, cumpre acrescentar a análise feita na obra de Somasundaram and Shrivastava [8]. Os autores afirmam que um SAD deve fornecer aos clientes o acesso direto a todo o sistema de arquivos, ao mesmo tempo em que deve garantir a gestão eficiente dos dados, além de sua segurança.

Novamente, se por um lado a facilidade de acesso provida pelos SAD é uma característica desejável (e por isso intrínseca), por outro há que se levar em conta aspectos ligados à segurança destes sistemas. Vários foram os autores que trataram a dicotomia segurança *versus* usabilidade, como publicado em [9], [10] e [11], apenas para citar alguns exemplos. Dentre eles, observa-se a convergência para um ponto em comum, inclusive pacificado pela literatura: segurança e usabilidade são conceitos inversamente proporcionais. Pode-se dizer, sem prejuízo do contexto e em outras palavras, que o aumento nos níveis do primeiro implica necessariamente diminuição no segundo.

Cabe neste ponto definir o conceito de usabilidade: é o grau de facilidade com que um usuário consegue interagir com determinada interface. Em outras palavras, define a simplicidade com que uma pessoa pode utilizar um sistema a fim de executar uma ação específica [12].

Esta não é uma ideia distante da desenvolvida por Yahya et al. [13]. No mesmo sentido, ponderam os autores que "um conjunto completo de controles pode não ser totalmente implementado devido a vários desafios, tais como diminuição da disponibilidade, menor conveniência do usuário, necessidade de uma infraestrutura robusta, dentre outros".

Para além das ameaças ligadas à autenticação, outros ataques contra a disponibilidade, integridade e confidencialidade dos dados armazenados em SAD também devem ser examinados. A disponibilidade dos dados pode ser comprometida por ataques de negação de serviço, conforme é possível observar na modelagem de ataques a Sistemas Distribuídos publicada em [14]; a integridade dos dados pode ser violada por meio de ataques de *ea-*

vesdropping[1], cujos padrões de abusos tanto de comunicação em rede quanto sobre dados em armazenamento em Sistemas Distribuídos foram classificados em uma extensa taxonomia demonstrada em [15]; a confidencialidade dos arquivos armazenados, cujo esforço se concentra em garantir que usuários não-autorizados sejam incapazes de decifrar dados para os quais não possuem a devida permissão, pode ser deturpada por conta da gerência inadequada da criptografia aplicada aos arquivos.

Incidentes que levem a falhas nas características aqui citadas podem resultar em perda de acesso ou mesmo na perda efetiva de dados, acarretando graves consequências para a organização que utilize tal tecnologia para compartilhamento de arquivos, pelo que se justifica a presente análise.

Em razão de todas essas nuances, este trabalho propõe a execução de um exame circunstanciado, sob o ponto de vista da segurança, de um Sistema de Arquivos Distribuídos em particular denominado `OpenAFS`[2]. Utilizado em diversas organizações[3] como suporte para trabalho colaborativo em escala global, o `OpenAFS` é um projeto de *software* de código aberto que implementa o protocolo AFS, usado para acessar arquivos em uma rede como se estivessem em um disco local, fornecendo uma arquitetura cliente-servidor para compartilhamento de arquivos e replicação de conteúdo distribuído [16].

Serão analisadas as características de segurança de uma implementação virtualizada do `OpenAFS`, com o propósito de avaliar as características de disponibilidade, integridade e confidencialidade dos dados armazenados, além de investigar o funcionamento do processo de autenticação tanto de usuários do sistema quanto de servidores entre si. Esta avaliação segue o padrão metodológico descrito na Seção 1.4 deste capítulo, de modo a proporcionar a sistematização dos resultados obtidos que viabilizou a análise efetuada na parte conclusiva do trabalho.

1.1 Organização do texto

Esta dissertação de mestrado foi dividida em cinco partes e, para atingir os objetivos propostos, foi elaborada da seguinte forma:

- inicia com a contextualização do trabalho, trazendo ao leitor a motivação, os objetivos e a metodologia utilizada na confecção do texto;
- passa ao Capítulo 2, que dedica-se à revisão da literatura relacionada aos principais conceitos teóricos ligados a Sistemas Distribuídos e Sistemas de Arquivos Distribuí-

[1] Em tráfego de rede, mensagens podem ser interceptadas, modificadas, substituídas, corrompidas ou simplesmente excluídas por um atacante. Já sobre dados armazenados, um invasor pode modificá-los, corrompê-los ou mesmo apagá-los.

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

[3] https://www.openafs.org/success.html

dos, por meio do exame de livros e diferentes fontes disponíveis nas principais bases de dados de artigos científicos;

– a seguir, o Capítulo 3 descreve o `OpenAFS`, apresentando seu histórico e discutindo em profundidade suas características de segurança, finalizando com uma comparação com outros Sistemas de Arquivos Distribuídos;

– o Capítulo 4, por sua vez, apresenta a análise da arquitetura de segurança do `OpenAFS`, revelando os resultados obtidos com a avaliação executada no ambiente de simulação; e

– finalmente, o Capítulo 5 apresenta as conclusões finais do da dissertação, expondo a colaboração do trabalho, assinalando os óbices encontrados no desenvolvimento das atividades e apontando oportunidades de trabalhos futuros.

1.2 Motivação

O desenvolvimento das tecnologias relacionadas a sistemas de armazenamento de dados em rede tem permitido, a par da complexidade da implementação, substanciais avanços na área de segurança dos dados em *storages* [17]. Esta segurança pode ser entendida como a aplicação de técnicas, políticas e monitoramento de parâmetros, tudo com o objetivo de garantir a disponibilidade, integridade e confidencialidade dos dados neles presentes [8].

Assegurar tais princípios não é tarefa apenas inscrita à temática de armazenamento de dados. Ao contrário, é objeto de padronizações internacionais na vasta área sobre segurança em computação. Diante deste cenário, cabe citar a ISO 27000:2018 [18], glossário cujo objetivo é padronizar termos e definições comumente utilizados na área de sistemas de gerenciamento de segurança da informação.

É importante definir, inicialmente, o termo Segurança da Informação (SI). Trata-se da aplicação e gerenciamento de controles que garantam os princípios da **confidencialidade**, da **integridade** e da **disponibilidade** da informação, considerando para isso o grande leque de ameaças existentes, de modo a assegurar a continuidade dos negócios e minimizar danos decorrentes de incidentes [18]. De acordo com o glossário, tais princípios são definidos como:

– **confidencialidade:** a informação não deve ser disponibilizada ou divulgada a indivíduos, entidades ou processos não-autorizados. De outra forma, este princípio garante a proteção da informação contra seu acesso ou sua divulgação não autorizada;

– **integridade**: a informação não deve ser alterada sem autorização, ou mesmo violada; e
– **disponibilidade**: a informação deve ser sempre acessível e utilizável a partir da demanda de um ente devidamente autorizado.

Soma-se a essa temática outra, que lhe é correlata. Organizações públicas e privadas lidam diariamente com crescentes volumes de dados. A administração e controle de tais dados massivos passa a ser fator crítico para o funcionamento dessas instituições. Dados como documentos, projetos, máquinas virtuais, enfim, arquivos de toda espécie são guardados em sistemas de armazenamento situados em *data centers* de alta criticidade[4], de tal sorte que o valor destes ativos é altamente significativo para as organizações.

Delineado este quadro, o presente trabalho buscou analisar uma solução de armazenamento de dados distribuído que faz frente aos desafios apresentados, baseada na utilização de *software* de código aberto denominada `OpenAFS`.

O `OpenAFS` é um Sistema de Arquivos Distribuídos projetado para fornecer acesso transparente a arquivos remotos, tendo como um dos principais objetivo a identificação da escalabilidade pela utilização de *cache* nos nós clientes [7]. Milicchio and Gehrke [19] definem, de modo mais pormenorizado, que o `OpenAFS` é um sistema altamente escalável, que permite administração e manutenção flexíveis, fornecendo ainda recursos importantes como replicação e *backup*. Com base nas palavras destes autores, o `OpenAFS` pode prover alto nível segurança aos dados armazenados com a utilização de tecnologias específicas que mantêm a consistência e a sincronia de todas as informações armazenadas entre seus bancos de dados distribuídos. Ainda segundo [19], somam-se a estes recursos outras características desejáveis em um sistema seguro, como tráfego cifrado de dados, proteção de autenticação, monitoramento de processos e capacidade de restabelecimento automático em caso de falhas.

Nesse passo, esta dissertação propõe uma análise minuciosa dos aspectos de segurança do `OpenAFS`, haja vista que a pesquisa bibliográfica executada inicialmente não revelou estudos que tratam especificamente deste tema. Esta é uma razão necessária e suficiente para justificar a escolha do `OpenAFS` como sistema alvo desta pesquisa. Mesmo que os estudos consultados forneçam informações acerca dos recursos de segurança disponíveis no `OpenAFS`, não foram consideradas hipóteses de

[4] A criticidade mensura o impacto ao negócio da organização em caso de falha total ou parcial de um ativo.

testes e análises detalhadas, pelo que se foi identificada a lacuna existente na literatura, apontando-se a necessidade de efetuar um estudo direcionado para preencher tal hiato.

1.3 Objetivos

Esta Seção cuida de apresentar os objetivos geral e específicos desta dissertação.

1.3.1 Objetivo geral

O propósito deste trabalho é o de proceder a uma análise da segurança de uma implementação de um Sistema de Arquivos Distribuídos operacionalizado com o `OpenAFS`, avaliando os quesitos de segurança ligados à disponibilidade, integridade e confidencialidade oferecidos pelo sistema, bem como a segurança de seu sistema de autenticação.

1.3.2 Objetivos específicos

Para que o objetivo geral seja alcançado, é necessário atingir os seguintes objetivos específicos:

1. apresentar a fundamentação teórica em forma de pesquisa bibliográfica, de modo a identificar as características dos Sistemas Distribuídos (SD) e, mais especificamente, dos Sistemas de Arquivos Distribuídos (SAD), verificando os aspectos de Segurança da Informação ligados a disponibilidade, integridade e confidencialidade dos dados manipulados pelos sistemas;
2. identificar diferentes tipos de SAD e comparar as características técnicas de segurança destes com o funcionamento do `OpenAFS`, a fim de particularizar a pesquisa com estudos que tratam especificamente do `OpenAFS`, com vistas a compreender os níveis de segurança que o sistema oferece aos dados nele armazenados;
3. executar testes com cada um dos subsistemas do `OpenAFS` que executam operações ligadas à disponibilidade, integridade, confidencialidade dos dados e de credenciais de acesso, avaliando os resultados por meio da sistematização dos dados coletados, de modo a permitir uma comparação com o estado da arte na área de segurança de dados em SAD.

1.4 Método

Método é uma palavra que deriva do termo grego *methodos*. Este termo pode ser traduzido como um "caminho para chegar a um fim". Vários autores abordam o método como ferramenta necessária para orientar o processo de pesquisa. Wazlawick [20], por exemplo, descreve o método como "a sequência de passos necessários para demonstrar que o objetivo proposto foi atingido, ou seja, se os passos definidos no método forem executados, os resultados obtidos deverão ser convincentes". Gil [21] complementa o raciocínio, apontando que a pesquisa é um "processo formal e sistemático de desenvolvimento do método científico". De acordo com o autor, "o objetivo fundamental da pesquisa é descobrir respostas para problemas mediante o emprego de procedimentos científicos".

Cabe supor, dos conceitos apresentados, que a rigorosa definição dos passos necessários para conduzir a pesquisa, de modo que os objetivos sejam alcançados, é fator necessário para a adequada estruturação de um trabalho de pesquisa.

Sinteticamente, a metodologia proposta neste trabalho visa, a partir da pesquisa bibliográfica executada no Capítulo 2 e com a compreensão detalhada do sistema obtida com o Capítulo 3, proceder a uma pesquisa quantitativa[5] que, por sua vez, permitirá a comparação do `OpenAFS` com outros SAD e com os mais recentes trabalhos da área de segurança. Desta forma, será possível mapear e categorizar o nível de segurança que o sistema pode conferir aos dados que armazena e manipula.

Nestas condições, esta Seção visa detalhar a série de fases planejadas para efetivar este trabalho, com a finalidade de se atingir os objetivos propostos na Seção 1.3.

1.4.1 Caracterização do problema

Um problema científico é, de modo elementar, uma questão ainda não totalmente resolvida que permanece sendo discutida em algum domínio do conhecimento. O trabalho em tela levantou que o `OpenAFS` possui recursos de segurança, mas não foi, até o momento, realizada uma pesquisa que o comparasse com as principais referências na área. Para fazer frente a esta lacuna, caracterizou-se o seguinte problema, aqui formulado como pergunta:

O `OpenAFS` fornece garantias para os princípios da segurança da informação (disponibilidade, integridade e confidencialidade) de dados nele armazenados?

[5] Tradução de alguns parâmetros em números que, após análise estatística, podem apontar padrões conclusivos e aplicáveis ao estudo.

1.4.2 Formulação da hipótese de pesquisa e justificativa

Uma hipótese, nas palavras de Gil [21], é uma proposição testável que pode vir a ser a solução de um problema. Levando-se em conta o apresentado na Seção 1.2, é cabível supor, neste ponto, que a questão colocada no item anterior tenha uma solução viável. Isso posto, propõe-se como modelo de análise a seguinte hipótese de pesquisa:

O `OpenAFS` apresenta-se como um Sistema de Arquivos Distribuídos que fornece recursos necessários e suficientes que garantam a segurança da informação (disponibilidade, integridade e confidencialidade) dos dados nele armazenados.

A definição desta hipótese utilizou-se do método hipotético-dedutivo tal como posto por Gerhardt and Silveira [22], uma vez que sua criação partiu das ideias conceituais apresentadas na fase inicial deste capítulo, as quais podem (porém, não necessariamente) explicar o objeto de estudo.

A propositura desta hipótese de pesquisa baseia-se no fato de que os conhecimentos disponíveis sobre o sistema apresentaram-se, até a fase de pesquisa, insuficientes para assegurar que o `OpenAFS` oferece recursos que garantam adequados níveis de segurança aos dados que o sistema manipula, uma vez que não foram identificadas análise de segurança deste SAD, fatores estes que justificam o presente trabalho.

1.4.3 Etapas e resultados esperados

Este estudo iniciou-se com a pesquisa bibliográfica a respeito dos SAD existentes, debruçando-se sobre as análises já existentes sobre os aspectos de segurança destes sistemas. Como poderá ser visto a partir do próximo capítulo, a fundamentação teórica visa fornecer o entendimento básico dos conceitos que serão usados na fase subsequente, na qual é executada a análise pormenorizada das características de segurança do `OpenAFS`.

Com base nessas premissas, este trabalho utilizou uma abordagem sistemática por meio de pesquisa quantitativa que possibilitou a coleta de dados intrínsecos do `OpenAFS`. A análise quantitativa foi necessária para que dela pudesse ser extraída uma coleção de dados que, então, pudessem ser formalmente modelados de tal forma que denotem a performance do sistema. Esta última fase, analítica, permitiu comparar o `OpenAFS` com outros SAD e, desta forma, concluir a respeito de seus níveis de segurança.

Desta forma, a fim de responder à questão-problema, foram definidos os seguintes passos para delinear a fase de pesquisa do trabalho:

- identificar o estado da arte na área de segurança em SAD;
- conhecer detalhadamente o `OpenAFS`, coletando dados e informações a respeito do sistema;
- comparar o `OpenAFS` com outros SAD e com o estado da arte na área de segurança; e
- analisar uma implementação `OpenAFS` visando identificar o alinhamento da solução com as garantias da segurança da informação, especificamente disponibilidade, integridade e confidencialidade.

A fase de análise de implementação, por sua vez, exige a execução de experimentos por meio dos quais são efetuados testes de segurança e desempenho. Propõe-se a adoção de um ambiente virtualizado utilizando `OpenAFS`, de modo que sejam executados os seguintes procedimentos:

1. Execução de testes de *Benchmark*[6] para coleta de *datasets*[7];
2. Registro formal dos dados;
3. Análise e interpretação dos dados de *Benchmark*; e
4. Apresentação das conclusões.

Desta análise são esperados os seguintes resultados:

- entendimento dos aspectos de segurança do sistema de autenticação utilizado pelo `OpenAFS`;
- compreensão dos sistemas relacionados ao controle de integridade dos dados manipulados pelo `OpenAFS`; e
- percepção dos recursos que viabilizam a disponibilidade dos dados no SAD.

[6] Análise de desempenho de determinado equipamento, com objetivo de mensurar sua performance.
[7] Conjunto de dados classificados por variáveis, coletados de sistemas ou equipamentos.

1.5 Considerações finais

Este capítulo apresentou o contexto em que se insere o trabalho e ofereceu ao leitor as condições de delinear o escopo da dissertação e de compreender a metodologia empregada na construção das diferentes fases da pesquisa elaborada.

A identificação dos objetivos geral e específicos teve como meta sintetizar o que se busca com este trabalho de pesquisa. Já a definição dos primeiros conceitos teóricos forneceram, ainda que prelimarmente, o embasamento necessário para entender as motivações da análise a ser apresentada.

A parte que se segue fornecerá uma fundamentação teórica mais extensa e profunda, cujos conceitos serão necessários ao entendimento das características de segurança do `OpenAFS`, abordadas na Seção 3 deste trabalho.

Capítulo 2

Fundamentação teórica

Este capítulo tem como objetivo estruturar conceitualmente o levantamento de publicações e do referencial teórico a respeito do assunto, de forma a sustentar o desenvolvimento da fase de pesquisa. Apresentar a revisão de literatura e determinar o "estado da arte" possibilita a identificação de lacunas existentes na área e justifica as fases posteriores do trabalho [22].

A fim de atingir estes propósitos, apresenta-se inicialmente uma revisão conceitual sobre Sistemas Distribuídos e caracterizam-se, posteriormente, os Sistemas de Arquivos Distribuídos. Ainda, são definidos formalmente os conceitos relacionados a persistência de dados, e, finalmente, discutido o estado da arte na área de segurança da informação, explorando-se os aspectos teóricos relacionados a esta temática no âmbito de Sistemas de Arquivos Distribuídos.

2.1 Sistemas Distribuídos (SD)

Não há uma definição única para Sistemas Distribuídos. De modo mais abrangente, Coulouris et al. [7] definem que um "Sistema Distribuído é aquele no qual os componentes de *hardware* ou *software*, localizados em computadores interligados em rede, se comunicam e coordenam suas ações apenas enviando mensagens entre si". Já Tanenbaum and Steen [23] estabelecem que SD formam "um conjunto de computadores independentes entre si que se apresenta a seus usuários como um sistema único e coerente".

Um SD passa a seus usuários a noção de que há apenas um sistema em atividade na rede. Esta abstração, chamada também de imagem única, conduz ao conceito de compartilhamento transparente de recursos. Tal concepção, requisito fundamental para os SD, omite dos usuários a complexidade das respectivas implementações e, consequentemente, facilita sua utilização.

Ainda de acordo com Tanenbaum and Steen [23], a implementação de Sistemas Distribuídos pode proporcionar os seguintes benefícios:

- transparência ao usuário: a utilização de múltiplos recursos não é observável ou sentida pelos utilizadores dos recursos disponíveis;
- economia: equipamentos específicos apresentam melhor custo/benefício;
- velocidade: um SD apresenta maior capacidade de processamento que equipamentos isolados;
- confiabilidade: não apresenta pontos únicos de falhas, e a ocorrência destas não interrompe o sistema como um todo [7]; e
- escalabilidade: apresenta condições de ampliação de seu estado, seja em processamento, armazenamento ou outros recursos do sistema.

Sob outra ótica, os autores afirmam que os SD podem apresentar certas desvantagens. Cabe citar, no contexto deste trabalho:

- complexidade de implementação, em comparação com sistemas centralizados;
- saturação de redes que não estejam adequadamente preparadas para o projeto; e
- a consistência dos dados pode ser comprometida por problemas de sincronização ou replicação entre os sistemas [24].

A essência dos Sistemas Distribuídos forma a base dos Sistemas de Arquivos Distribuídos, os quais são discutidos na próxima seção.

2.2 Sistemas de Arquivos Distribuídos (SAD)

Sistemas de Arquivos Distribuídos são casos particulares de Sistemas Distribuídos. Conforme Coulouris et al. [7], um sistema como este "permite aos programas armazenarem e acessarem arquivos remotos exatamente como se fossem locais, possibilitando que os usuários acessem arquivos a partir de qualquer computador em uma rede".

Esta explicação, fornecida na mesma linha por Mullender [25], mostra, em outras palavras, que os SAD oferecem funcionalidades similares a um sistema de arquivos tradicional, porém sendo executado em um ambiente distribuído. De acordo com os autores, SAD devem permitir que vários processos compartilhem dados por longos períodos. Além disso, fatores como desempenho e segurança no acesso aos arquivos armazenados em um sistema remoto devem ser compatíveis aos arquivos armazenados em discos locais.

Da mesma forma que os SD, os SAD também apresentam vantagens em sua implementação. Podem ser citadas, dentre outras:

- flexibilidade: o acesso aos arquivos independe da localização do cliente;
- transparência: a complexidade dos processos envolvidos não é percebida pelos usuários;
- escalabilidade: o sistema suporta aumento de recursos;
- confiabilidade: permite redução de danos em arquivos, ou mesmo perdas; e
- integridade: gerencia simultaneidade de acessos a um mesmo arquivo.

De igual modo, anomalias indesejáveis também podem ocorrer, como por exemplo:

- problemas de desempenho: acessos remotos normalmente são mais lentos, se comparados a acessos locais;
- maior número de causas de indisponibilidade: falhas em componentes externos (rede, por exemplo), podem interferir em seu funcionamento; e
- problemas de acesso: enquanto um arquivo é acessado por um cliente, outros clientes não podem modificá-lo[1].

Vale ressaltar, conforme introduzido no começo deste trabalho, que a facilidade de acesso a dados proporcionada por SAD pode dificultar o processo de garantia da segurança dos dados, em especial no tocante à confidencialidade. Sobre isso, cabe introduzir que as questões ligadas à segurança em Sistemas de Arquivos Distribuídos podem ser relacionadas ao arquivo propriamente dito ou ao usuário.

Com relação ao primeiro caso, a adoção de técnicas de criptografia fazem frente aos desafios, cujos métodos aplicáveis serão discutidos na Subseção 2.5.2. Esta Subseção aborda, também, soluções para o segundo caso, relacionadas, por sua vez, ao controle de acesso.

A próxima Seção trata de um sistema específico para compartilhamento de arquivos denominado *Network-Attached Storage (NAS)* (Armazenamento Conectado à Rede), cuja função é semelhante à dos SAD, porém com características distintas.

2.3 Network-Attached Storage (NAS)

Network-Attached Storage (NAS), também conhecido como "Armazenamento Conectado à Rede", é um dispositivo IP com sistema operacional embarcado que integra o *hardware*

[1]Esta ação de bloqueio é chamada de *lock*.

ao serviço de armazenamento de dados, projetado especificamente para compartilhamento de arquivos [26].

Atuando com outros dispositivos na rede, o equipamento assume exclusivamente a função de processamento do sistema de arquivos, deixando os demais servidores com poder de processamento livre para manipular conexões e operar em suas funções específicas.

Como consequência de sua especificidade, dispositivos NAS oferecem melhor desempenho na oferta de serviços de arquivos se comparados a servidores de uso comum, sendo extensivamente utilizados nas redes das organizações [8]. Entre os benefícios de sua utilização podem ser citados:

- eficiência: seu sistema operacional é especializado em compartilhar arquivos;
- flexibilidade: permite compatibilidade com diferentes tipos de clientes e protocolos;
- centralização do serviço: em um único sistema é possível gerenciar todos os dados armazenados;
- alta disponibilidade: possuem redundância de componentes que permite implementação de *Failover Clusters*[2], além de recuperação e replicação de arquivos.

Com relação aos óbices associados a implementações de NAS, podem ser citados:

- necessidade de otimização dos *storages* para as especificidades organizacionais;
- indispensável acompanhamento de atualizações de *drivers* e Sistema Operacional; e
- dependência de infraestrutura de rede adequada e resiliente.

Conforme Pawar et al. [27], dispositivos NAS são projetados de tal forma que permitam a qualquer usuário legítimo contactar algum dos nós disponíveis para reconstruir o arquivo demandado, que pode estar segmentado pelos demais entes do sistema. Com isso, da mesma forma que nos SD e, em última análise, nos SAD, a garantia da segurança dos dados armazenados pode ser dificultada pela facilidade de acesso aos sistemas.

Neste passo, Kumar [28] afirma que "embora a implantação seja relativamente simples, as organizações devem ter o cuidado de garantir que níveis adequados de segurança para os arquivos sejam fornecidos durante a configuração dos dispositivos NAS". É, também, o que alertam Somasundaram and Shrivastava [8]: "sistemas de armazenamento são expostos a várias ameaças à segurança das informações neles guardadas, o que pode afetar dados críticos das organizações e interromper serviços essenciais".

[2]Dois ou mais equipamentos que trabalham em conjunto, de maneira que uma eventual falha em um dos participantes do *cluster* não impacte na disponibilidade do todo.

Mais uma vez foi possível identificar a separação entre segurança *versus* usabilidade no âmbito de compartilhamento de arquivos em rede. Encontrar o equilíbrio coerente entre essas duas importantes funções é um desafio constante, e a base teórica para identificar os efeitos positivos desta harmonia no campo dos Sistemas de Arquivos Distribuídos depende da compreensão de seus requisitos, os quais são apresentados a seguir.

2.4 Requisitos dos Sistemas de Arquivos Distribuídos

Antes de explicitar os requisitos dos SAD, é imperativo complementar a compreensão de Coulouris et al. [7], citada preliminarmente na Seção 2.2. Os autores tratam a respeito de compartilhamento de informações armazenadas em redes locais e *intranets* e, conforme suas palavras "um serviço de arquivos permite que os programas armazenem e acessem arquivos remotos exatamente como se fossem locais, possibilitando que os usuários acessem seus arquivos a partir de qualquer computador."

Cabe acrescentar a este pensamento a afirmação colocada em sequência. Conforme os autores, a utilização de SAD pode "reduzir a necessidade de armazenamento local, facilitando o gerenciamento e execução de cópias de segurança de uma organização". Chamam a atenção, ainda, para o fato de que o serviço de compartilhamento de arquivos é o mais usado em redes organizacionais e, em decorrência, sua funcionalidade e desempenho são críticos.

Nessa linha de raciocínio, é possível afirmar que um Sistema de Arquivos Distribuídos pode assumir função crítica em redes organizacionais. Compreender as particularidades destes sistemas é primordial para a análise que se pretende executar com esta dissertação. A fim de alcançar esta compreensão, a seguir são estudados os requisitos relacionados com os SAD.

2.4.1 Transparência

Coulouris et al. [7] assim definem transparência: "a ocultação, para um usuário final ou para um programador de aplicativos, da separação dos componentes em um sistema distribuído de modo que o sistema seja percebido como um todo, em vez de uma coleção de componentes independentes".

Tal qual introduzido nas seções 2.1 e 2.2, e dada a importância deste conceito, cabe destacar como Tanenbaum and Steen [23] subdividiram a transparência, a saber:

- transparência no acesso: permite que recursos locais e remotos sejam acessados com o uso de operações idênticas;

– transparência na localização: permite que os recursos sejam utilizados sem o conhecimento de sua localização física;

– transparência na migração: os recursos podem ser movidos para outros locais;

– transparência na relocação: permite que migrações não afetem a operação de usuários ou aplicações;

– transparência na replicação: permite que os recursos sejam replicados para aumentar a confiabilidade e o desempenho do sistema, sem o conhecimento dos usuários ou suas aplicações;

– transparência na concorrência: permite que múltiplos usuários compartilhem recursos automaticamente; e

– transparência na falha: permite que eventuais panes no sistema não sejam sentidas pelos usuários.

O conceito de transparência, intrínseco dos Sistemas Distribuídos, é também um dos atributos do `OpenAFS`. Conforme Milicchio and Gehrke [19], este é um Sistema de Arquivos Distribuídos transparente ao usuário, que oferece a possibilidade de acesso a arquivos independente da localização ou do sistema operacional do cliente. O `OpenAFS` será estudado em detalhes no próximo capítulo.

2.4.2 Detecção e tolerância à falhas

A natureza do projeto de Sistemas Distribuídos leva naturalmente à percepção de relações de interdependência entre seus vários componentes. São estes encadeamentos que tornam o todo passível de problemas inevitáveis, como erros, falhas ou mal funcionamento. Estes eventos podem causar problemas de acesso ou instabilidade na disponibilidade dos serviços providos pelo conjunto.

Neste contexto, é relevante perceber a diferença entre os termos **erro** e **falha**, pois este decorre daquele, não sendo portanto sinônimos, como pode parecer à primeira vista. De acordo com Delamaro et al. [29], erros ocorrem durante a execução de um programa, e são caracterizados por um estado inconsistente ou inesperado, como por exemplo uma mensagem corrompida. Já as falhas são comportamentos inesperados de uma aplicação ou programa, sendo causadas pelos erros.

Tanenbaum and Steen [23] definem um esquema que tipifica os processos de falhas em SD:

– *crash failure*: falhas relacionadas a desligamentos de servidores que até então operavam normalmente;

– *omission failure*: falhas relacionadas à omissão no envio e recebimento de mensagens;

– *timing failure*: falhas relacionadas a atrasos no tempo de resposta;

– *response failure*: falhas relacionadas ao formato correto de respostas; e

– *arbitrary failure*: relacionada a *crash failure*, é uma combinação das duas anteriores, apresentando problemas de respostas em tempos aleatoriamente atrasados.

Para além dos tipos citados, é relevante ressaltar outra espécie de falhas. Denominadas de arbitrárias ou bizantinas [30], tais falhas são resultados de respostas fora do padrão do protocolo utilizado pelo sistema do servidor. Essa atipicidade normalmente ocasiona problemas na detecção das falhas geradoras das respostas, ou até mesmo a impossibilidade da detecção. As falhas bizantinas tem, no contexto deste trabalho, importância significativa, pois podem ser causa de problemas de segurança.

Um processo bizantino tem a possibilidade de se passar por outro processo e iniciar a transiimissão de mensagens com valores errados, duplicados ou até mesmo não enviar as mensagens no padrão que o protocolo da aplicação define. Portanto, a tolerância à falhas bizantinas é importante no contexto da segurança, especialmente no campo dos Sistemas Distribuídos.

Evitar que tais eventos se transformem em erros que venham a prejudicar o bom funcionamento do sistema como um todo é o que se define como Tolerância à Falhas. Neste enfoque, e dada a importância do tema, pesquisadores como Gupta and Saini [31] propuseram técnicas para incrementar a segurança e a tolerância à falhas em Sistemas de Arquivos Distribuídos, abordando a aplicação de múltiplos níveis de tolerância à falhas, de modo a conferir segurança no armazenamento de arquivos, com transparência e independência de localização.

Como esperado, as ações para controlar as falhas em um sistema tem como consequência o aumento em sua disponibilidade. Por outro lado, o trabalho de Ostovari and Wu [32] revela uma outra consequência, desta vez indesejável: mais redundância implica aumento na vulnerabilidade a ataques de *eavesdropping* em sistemas de armazenamento. Este trabalho, bem como a análise de suas repercussões sobre a segurança em Sistemas de Arquivos Distribuídos, serão apresentados a seguir.

Finalmente, tempo de inatividade em sistemas de armazenamento de uso intensivo também implica em custos, uma vez que nos casos de solução de continuidade, sistemas críticos *offline* acarretam atraso de prazos e problemas na área operacional dos envolvidos. Para minimizar os efeitos negativos da ocorrência de falhas é empregada, normalmente, a técnica de replicação de dados, cujos fundamentos são explanados na próxima Subseção.

2.4.3 Replicação

Um dos conceitos utilizados para fornecer confiabilidade a um Sistema de Arquivos Distribuídos é a replicação. Um arquivo pode ser representado por várias cópias de seu conteúdo em diferentes pontos, de modo que, em caso de falha de um servidor, um outro possa oferecer a um cliente uma cópia daquele arquivo [7].

Se por um lado a replicação de arquivos aumenta a confiabilidade de um SAD, por outro, a existência de múltiplas cópias pode acarretar problemas de consistência, já que se uma cópia é modificada, torna-se diferente das demais, fato que leva à necessidade de modificações em todas as outras cópias, de modo a garantir a consistência dos dados [7].

Para fazer frente a este problema, são necessários serviços específicos que realizem a monitoração de processos e garantam, com isso, a sincronização de seus dados. Neste sentido, o `OpenAFS` utiliza uma tecnologia própria para manter a consistência entre suas bases de informações, e suas características particulares serão estudadas no Capítulo 3.

A replicação é, ainda, uma das muitas implementações possíveis para cópias de segurança *(backups)*, uma vez que diminui a probabilidade de perda efetiva de dados. O `OpenAFS` oferece a capacidade de replicar volumes em servidores de arquivos. Usando este recurso de forma sincronizada e automatizada, as réplicas podem se tornar uma forma efetiva de cópia de segurança [19]. A análise detalhada desta característica será apresentada na Seção 3.4.

2.4.4 *Cache*

De modo geral, *cache* é um serviço que disponibiliza, em uma área próxima aos clientes, os dados por eles recentemente usados. Desta forma, se um determinado dado é requisitado por um outro processo cliente, o serviço tem condições de fornecê-lo mais rapidamente, caso possua uma cópia atualizada. Caso contrário, o *cache* atualiza o dado solicitado e o entrega, também guardando-o para um provável uso subsequente.

Exemplos clássicos deste recurso são os *caches* de *proxies*[3] de acesso à *Internet*, largamente utilizados em redes corporativas para otimizar o tráfego *Web*. Os *proxies* aceitam solicitações de clientes locais e as repassam para servidores da *Web* e, recebidas as respostas, estas são repassadas aos clientes. A vantagem do *proxy* é poder armazenar os resultados em *cache* e reaproveitá-los para outro cliente, se necessário [23].

Entretanto, não apenas neste cenário a utilização do *cache* é vantajosa. Este recurso é, do mesmo modo, largamente utilizado em Sistemas de Arquivos Distribuídos. Nestes ambientes o *cache* pode ser armazenado tanto no lado do cliente, quanto no lado do ser-

[3]Servidor posicionado entre uma aplicação cliente e um servidor final, que funciona como intermediário entre os dois entes, interceptando as solicitações do primeiro para o segundo e controlando o fluxo de dados entre eles.

vidor. No primeiro caso, o cliente armazena conteúdo que foi acessado por um usuário que está executando uma aplicação. No segundo, o servidor guarda dados que foram solicitados pela maioria dos usuários, já que tais dados podem ser potencialmente utilizados por algum utilizador no futuro [33]. A Figura 2.1 ilustra a utilização do *cache* no segundo caso:

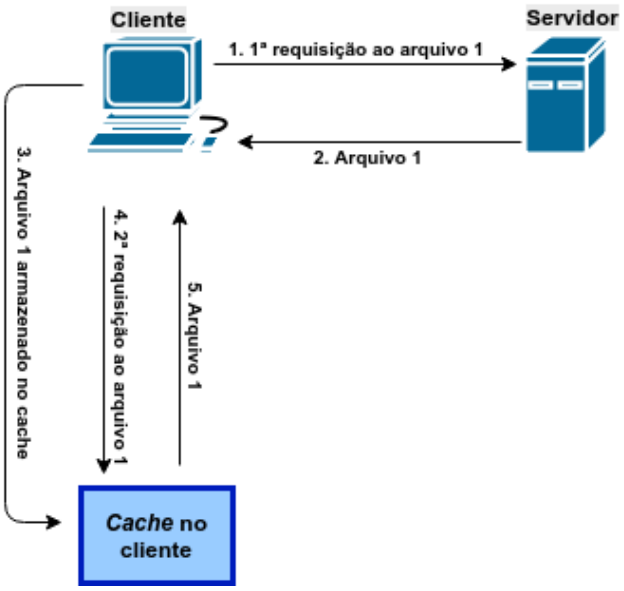

Figura 2.1: Funcionamento do *cache* no cliente.

O armazenamento local, i.e, no lado do cliente, objetiva diminuir tanto o tráfego de rede quanto o processamento resultante de reiterados acessos a um mesmo arquivo, resultando em melhora global no desempenho e em maior velocidade de acesso. O `OpenAFS` utiliza esta abordagem do *cache*.

Quando uma aplicação cliente solicita um arquivo, um processo envia uma solicitação para o servidor e armazena uma cópia deste em disco ou memória, de modo transparente ao usuário. Em outras palavras, o cliente não tem a necessidade de saber onde o arquivo se encontra, conforme citado na Seção 2.4.1.

O funcionamento detalhado do *cache* no `OpenAFS`, bem como os processos usados para garantir que o armazenamento local esteja atualizado com a última versão do arquivo, serão apresentados na Seção 3.1.2.

2.5 Segurança

A demanda por sistemas de armazenamento seguros e confiáveis tem aumentando muito atualmente. Esta necessidade está relacionada à tendência mundial de aumento no nú-

mero de violações de dados em organizações ao redor do planeta, como se pode verificar no relatório *"2018 Cost of a Data Breach Study: Global Overview"* [34], publicado recentemente pela IBM *Security*.

O documento, elaborado com os dados de mais de 2.200 profissionais de TI e com informações de quase 500 empresas que sofreram violações em dados organizacionais a cada ano, aponta que o custo total médio de violação de dados no mundo subiu de USD 3,62 milhões em 2017 para USD 3,86 milhões em 2018, o que corresponde a um aumento médio de 6,4%.

O Brasil encontra-se entre os 15 países pesquisados. De acordo com o relatório, apesar de possuirem um custo ligado à violação de dados menor do que outros países, as organizações brasileiras estão mais propensas a sofrer novas violações, com probabilidade estimada da ordem de 43%, seguido pela África do Sul e França, com estimativas de 40,9% e 35,1%, respectivamente.

Tanenbaum and Steen [23] também chamam a atenção para a questão da segurança em Sistemas Distribuídos: "Se por um lado os processos devem ser desvinculados e independentes, por outro é necessário garantir integridade e confidencialidade dos dados".

A título de complemento, vale citar que, de acordo com Modi et al. [35], o aumento de casos de acesso mal-intencionado ou não autorizado a informações armazenadas remete à necessidade de que os Sistemas de Arquivos Distribuídos devem estar preparados para lidar com a segurança de dados de forma eficaz e transparente, sobretudo quando os dados estão expostos a um ambiente não confiável.

Sob todos os ângulos enfocados, conclui-se que há real necessidade de se investigar os requisitos de segurança e as ameaças às quais um dado Sistema de Arquivos Distribuídos está exposto, de modo a definir o nível de segurança que a solução oferece aos dados armazenados.

2.5.1 Segurança no armazenamento

Antes de se discutir a segurança em Sistemas de Arquivos Distribuídos, há que se pontuar inicialmente os atributos genéricos relacionados aos sistemas de armazenamento. Conforme Somasundaram and Shrivastava [8], são quatro os atributos primários para a segurança em sistemas de armazenamento em geral:

- Disponibilidade: "garantia de acesso oportuno aos usuários , independente de falhas isoladas";
- Integridade: "garantia de que a informação não seja indevidamente alterada, detectando e protegendo contra mudanças não autorizadas";

– Confidencialidade: “fornece a segregação necessária para a informação e garante que apenas os usuários autorizados tenham acesso aos dados”; e

– *Accountability*: “refere-se à contabilização de todos os eventos e operações que ocorrem na infraestrutura do *data center*, por meio da manutenção de registros de eventos que podem ser auditados ou rastreados posteriormente para o propósito de segurança”. Em outras palavras, pode ser definida como a capacidade de imputar responsabilidade a determinado ente, tendo como argumentos os registros de suas ações. A propriedade decorrente desta, ligada ao usuário, é definida como “Não-Repúdio”, que assegura a impossibilidade de se questionar a autoria de uma dada ação registrada em sistemas de informação pela *accountability*.

Daí se compreende, *a priori*, que a segurança em Sistemas de Arquivos Distribuídos deriva da segurança no armazenamento, que é genérica, e passa necessariamente pela garantia dos princípios da Segurança da Informação (Seção 1.2), porém com as particularidades que lhe são peculiares.

A próxima subseção cuida de apresentar os principais estudos relacionados, além de evidenciar e fundamentar a justificativa da hipótese de pesquisa.

2.5.2 Segurança em Sistemas de Arquivos Distribuídos

Esta subseção apresenta a pesquisa bibliográfica dos trabalhos na área de segurança em SAD e discute as propriedades básicas da Segurança da Informação com enfoque nos mecanismos de segurança em SAD. Como o foco do presente estudo é a análise de um Sistema de Arquivos Distribuídos específico, resta necessário discorrer sobre o estado da arte relacionado aos aspectos de segurança citados anteriormente.

A literatura atual fornece um grande número de propostas de solução para as mais diversas ameaças à segurança em Sistemas de Arquivos Distribuídos. Como ponto de partida e contextualização, cabe citar Grawinkel et al. [36]. Conforme os autores, se por um lado aspectos como tamanho dos arquivos, latência, largura de banda e até mesmo eficiência dependem principalmente do *hardware*, por outro, aspectos importantes relacionados à segurança, como confidencialidade, integridade e disponibilidade dos dados podem ser afetados pelo *software* que os utilizam.

Graf and Wolthusen [37] caracterizam as diferentes ameaças a um sistema de armazenamento, dividindo-as entre em ameaças *on-line* e *off-line*. Segundo os autores, ameaças *on-line* são definidas como as que ocorrem enquanto o sistema operacional executa a intermediação para o acesso aos recursos de armazenamento, e *off-line* quando essa mediação não é realizada.

Choi et al. [38] discutem o problema de intrusão em Sistemas de Armazenamento Distribuídos em situações em que pelo menos um dos nós do sistema apresenta problemas de segurança. De modo geral, os autores classificam tentativas de intrusão em dois tipos: **intrusão ativa**, por meio da qual o atacante consegue acesso em nível de modificação de arquivos, e **intrusão passiva** (*eavesdropping*), que permite tão somente a leitura de dados. Apresentam, então, uma análise de ambos os tipos de atacantes e oferecem um modelo referencial teórico para a capacidade de manutenção da confidencialidade em Sistemas de Armazenamento Distribuídos contra tais ações maliciosas.

Bžoch and Šafařík [24] também discorrem sobre a segurança em SAD: "a tendência atual para obtenção de segurança em Sistemas de Arquivos Distribuídos pode ser dividida em duas partes: comunicação segura em rede e segurança no armazenamento de arquivos". Ainda de acordo com os autores, "a comunicação segura em rede protege os dados contra ataques do tipo *eavesdropping* por meio de aplicação de criptografia", ao passo que "a segurança em sistemas de arquivos distribuídos tradicionais é resolvida usando autenticação e controlando o acesso aos arquivos".

Conforme já comentado, ataques do *eavesdropping* interceptam e monitoram de modo não-autorizado o tráfego de dados entre cliente e servidor. O objetivo deste abuso é capturar dados sensíveis (como credenciais de acesso, por exemplo) ou mesmo remontar o tráfego de dados na tentativa de recompor arquivos a partir da junção lógica de fragmentos de dados.

Nessa linha, Ostovari and Wu [32] abordam uma interessante relação entre aumento na redundância de *hardware*[4] e ataques de *eavesdropping*. De acordo com os autores, à medida que se aumenta a redundância, o sistema de armazenamento torna-se mais robusto contra falhas, mas, em contrapartida, um efeito indesejável ocorre: há um aumento na vulnerabilidade do sistema de armazenamento contra estes tipos de ataques.

A preocupação com a ocorrência de falhas é decorrência natural do funcionamento de *drives*, assim como outros meios de armazenamento de dados. Estes apresentam ciclo de vida útil que termina com o desgaste de suas mídias. Assim, com o objetivo de prevenir perdas de dados, há necessidade de se introduzir redundância, o que acarreta maior disponibilidade, uma vez que os arquivos estão armazenados em vários locais.

A solução proposta pelos pesquisadores aborda a codificação linear dos dados, utiliza programação linear para reduzir a complexidade da otimização da programação e trata tanto a probabilidade de ocorrência de falhas em Sistemas de Armazenamento Distribuídos, como a probabilidade de um usuário malicioso executar um ataque de *eavesdropping* em uma rede de armazenamento.

[4]Técnica que visa ao mascaramento de falhas em Sistemas Distribuídos, usando *hardware* adicional.

Sobre este mesmo assunto, cabe destacar o trabalho de Tian et al. [39]. Nele os autores reiteram a importância e a necessidade de procedimentos de replicação dos dados em ambientes compostos por servidores eventualmente vulneráveis. Com o objetivo de aumentar os níveis de segurança e confiabilidade em sistemas de arquivos distribuídos sem afetar a questão de desempenho, o artigo apresenta a integração de técnicas de replicação e fragmentação de dados[5] para os nós do sistema.

Conforme citado preliminarmente na Seção 2.5.1 e de acordo com Yumerefendi and Chase [40], "um sistema é considerado *accountable* se fornece um meio para detectar e expor comportamentos inadequados de seus utilizadores. Esta ferramenta proporciona incentivos para a cooperação e desencoraja comportamentos incorretos e maliciosos". Estes autores apresentam um conjunto de ferramentas que incorporam funções de gerenciamento de estado, armazenando evidências de execuções no sistema de armazenamento, que permitem ações de auditoria e processos de responsabilização. Além disso, atua no controle de acesso por meio de Access Control Lists (ACL) como parte do processo de avaliação da responsabilização. Implementada como serviço, a ferramenta armazena estados para comprovar evidências de violação de políticas de segurança.

A necessidade de implementação de controles de acesso é endossada também por outros autores, como é possível verificar em Bžoch and Šafařík [24], que ressaltam a necessidade de implementação de mecanismos de controle sobre privilégios de usuários e aplicação de criptografia para garantir a segurança nos Sistemas Distribuídos; e Lizhong and Huibo [41], que apresentam um esquema de detecção de intrusão específico para Sistemas de Armazenamento, baseado em agentes estáticos e dinâmicos, que produzem alertas de ataques que não impactam significativamente no tráfego da rede.

Pawar et al. [27] definem capacidades de sigilo e resiliência de um sistema de armazenamento distribuído como a "quantidade máxima de informações que o sistema pode armazenar com segurança, respectivamente, na presença de um um espião ou um adversário malicioso". Uma vez que os sistemas de armazenamento distribuídos naturalmente sofrem processos de falhas ([42] e [43]), aqueles autores focaram em mensurar a tolerância a estes eventos usando um modelo matemático. Este modelo pode fornecer limites superiores e inferiores relativos à capacidade que um Sistema de Arquivos Distribuídos pode apresentar para tolerar ataques passivos durante o processo de reparação em panes. Isso pode ocorrer porque ataques passivos durante a recuperação de falhas podem coletar dados de um novo nó no processo de reparo, inclusive conseguindo introduzir mensagens de erro.

Essa possibilidade necessariamente deve ser levada em consideração, uma vez que,

[5]Técnica de segurança que divide um arquivo sensível em vários fragmentos que por sua vez são distribuídos em diferentes servidores em um sistema de armazenamento distribuído.

conforme já citado, a ocorrência de falhas é intrínseca ao funcionamento dos SD, e ataques *eavesdropping* podem tomar proveito dessa característica para obter dados sensíveis no processo de reparação de falhas.

Como não poderia deixar de ser, a confiabilidade na transferência de dados em Sistemas de Arquivos Distribuídos é igualmente importante. Palacios et al. [44] propõem uma implementação de um protocolo que oferece suporte *multicast*[6] para enviar dados de forma confiável para diferentes grupos de nós de armazenamento em Sistemas de Arquivos Distribuídos. Transferências usando *multicast* permitem realizar a difusão simultânea e múltipla de dados necessários para oferecer redundância, provendo alta disponibilidade e tolerância a erros. De acordo com o texto, "uma das principais vantagens de se usar *multicast* em ambientes *clusterizados* é a melhora na transferência de dados quando do envio para múltiplos nós, de modo a reduzir significativamente a sobrecarga da rede".

2.6 Criptografia

Esta seção evoca os principais conceitos teóricos de criptografia, com especial atenção aos protocolos e algoritmos utilizados pelos sistemas que serão analisados no Capítulo 4.

O termo criptografia, em Ciência da Computação, pode ser entendido como a aplicação de técnicas derivadas de conceitos matemáticos, denominados algoritmos, usados para transformar mensagens em claro em mensagens ininteligíveis a quem não possui a informação correta para decifrá-las.

Conforme Stallings [45], os algoritmos e protocolos de criptografia podem ser agrupados em quatro principais áreas:

- Encriptação ou criptografia simétrica: utilizada para ocultar o conteúdo dos blocos ou fluxos contínuos de dados de qualquer tamanho;
- Encriptação ou criptografia assimétrica: usada para ocultar pequenos blocos de dados, como valores de função de *hash* e chaves de encriptação;
- Algoritmos de integridade de dados: empregados para identificar possíveis alterações em blocos de dados; e
- Protocolos de autenticação: esquemas baseados no uso de algoritmos criptográficos, desenvolvidos para autenticar a identidade de entidades.

O papel da criptografia está diretamente ligado aos conceitos de segurança em Sistemas de Arquivos Distribuídos, conforme comentado na Subseção 2.5.2. Isso posto, e tendo

[6] Transmissão simultânea de dados, direcionada de uma origem para um grupo de destinos.

em vista que a apresentação destes conceitos é relevante para o entendimento dos mecanismos apresentados nos próximos capítulos, é pertinente citar cada um deles, de modo a fundamentar a apresentação do funcionamento dos sistemas analisados nesta dissertação.

2.6.1 Criptografia simétrica e assimétrica

A criptografia simétrica, também chamada de criptografia de chave secreta ou de chave simples, consiste na aplicação de um segredo a um texto de uma mensagem, com o objetivo de transformá-lo em um outro texto cifrado. Este segredo pode ser um número, uma senha ou uma sequência de caracteres aleatórios, em forma de chave, e deve ser conhecido tanto por quem cifra quanto por quem decifra a mensagem. O inconveniente desta técnica é simples de se observar: como qualquer pessoa que conheça a chave é capaz de decifrar a mensagem que foi cifrada por ela, a troca, controle e distribuição destas chaves passam a ser um problema que pode impactar na segurança do processo.

A criptografia assimétrica, conhecida como criptografia de chave pública, traz uma solução para este problema, já que se utiliza de um par de chaves relacionadas entre si, de modo que uma delas pode ser distribuída publicamente e a outra permanece em segredo com o proprietário. Por este motivo, a primeira chave é denominada chave pública e a segunda, chave privada. Qualquer mensagem cifrada com a chave pública somente pode ser decifrada utilizando-se o mesmo algoritmo utilizado e usando sua respectiva chave privada. Com isso, o problema da distribuição de chaves existente no esquema simétrico é resolvido.

Para além dessas diferenças, vale citar que a criptografia de chave pública apresenta um custo relativamente maior de processamento e, por isso, é mais lenta se comparada com a criptografia simétrica. Em decorrência, a primeira é indicada para cifrar mensagens de tamanho reduzido, enquanto que a criptografia de chave simples é recomendada para criptografar arquivos maiores.

Como cada um dos tipos citados possui suas próprias vantagens, há a possibilidade de se usar um esquema híbrido que emprega as duas técnicas, aproveitando-se dos benefícios de ambas. Pode-se citar com exemplo a utilização de criptografia assimétrica para criptografar chaves de sessão, que são simétricas. Neste esquema, a criptografia de chave pública garante a distribuição segura de uma chave privada que, então, pode ser usada para cifrar arquivos maiores. Podem ser citados, como exemplos práticos deste método, os protocolos Transport Layer Security (TLS) [46] e Secure Shell (SSH) [47], dentre outros.

2.6.2 Funções de *hash*

De modo geral, uma função de *hash* é um cálculo matemático que opera dados de tamanho variável, tendo como resultado uma sequência de dados de tamanho fixo, comumente tratado como *checksum*. Uma importante característica desta função é a de que qualquer alteração em seu domínio resulta em grande mudança em sua imagem.

Tipos particulares de funções de *hash* são as denominadas funções de *hash* criptográficas. Funções desta natureza, de acordo com Stallings [45], são algoritmos para os quais é computacionalmente inviável: descobrir um objeto de dados que seja mapeado para um resultado de *hash* pré-especificado; ou identificar dois objetos diferentes de dados que sejam mapeados para o mesmo resultado de *hash*. Estas características fazem com que esta função apresente duas propriedades importantes: é uma função de via única e é, também, livre de colisões.

Em consequência, via de regra, o resultado de uma função de *hash* é uma informação que pode garantir a integridade dos dados de entrada, uma vez que uma mudança em qualquer parte da entrada desencadeia uma grande mudança em sua saída.

2.6.3 Protocolos de autenticação

A garantia da confidencialidade, propriedade que restringe o acesso à informação exclusivamente aos entes legítimos, conforme definido na Subseção 1.2, é uma imposição necessária em todos os contextos da segurança da informação em ambientes computacionais. Desta forma, o processo de autenticação de entidades, sejam elas pessoas ou equipamentos, deve seguir as melhores práticas de segurança de modo a garantir a validade das identidades alegadas.

Neste sentido, Stallings [45] apresenta dois tipos de protocolos de autenticação que buscam garantir a confidencialidade:

- De autenticação mútua, no qual as partes em comunicação se satisfazem mutuamente a respeito da identidade um do outro, trocando suas chaves de sessão, usando uma hierarquia de dois níveis de chaves de encriptação simétricas, usada para fornecer confidencialidade para a comunicação em ambientes distribuídos; e
- De autenticação de mão única, o qual exige que o emissor emita uma solicitação para o destinatário desejado, espere uma resposta que inclua uma chave de sessão e somente então envie a mensagem.

Ainda de acordo com o autor, um dos principais serviços de autenticação é o `Kerberos`. Discutido em detalhes nos Capítulos 3 e 4 deste trabalho, o sistema utiliza protocolos de

autenticação para operação em ambientes distribuídos abertos. Os usuários devem acessar, nestes ambientes, recursos em servidores distribuídos pela rede, mas os servidores devem restringir o acesso unicamente a usuários autorizados. Além disso, o `Kerberos` deve autenticar as solicitações de serviço em um ambiente em que uma estação de trabalho pode não ser confiável para identificar seus usuários corretamente com os serviços disponibilizados.

2.7 Considerações finais

Este capítulo apresentou uma visão geral sobre os Sistemas Distribuídos, particularizando os Sistemas de Arquivos Distribuídos e as questões relacionadas à segurança nestes sistemas. A análise dos mais recentes estudos ligados à segurança em Sistemas de Arquivos Distribuídos apontou que há um grande número de ameaças e que estas impactam diretamente no funcionamento dos sistemas de organizações.

Restou claro que os ataques do tipo *eavesdropping* são comumente observados contra fluxos de dados em Sistemas de Arquivos Distribuídos e, por conta disso, são largamente estudados na literatura. Por este motivo este tipo de ataque deve ser analisado criteriosamente neste estudo.

Verificou-se, ainda, que a possibilidade de implementação de diferentes recursos de segurança e controle podem servir para alavancar não somente a qualidade de acesso, mas principalmente aprimorar os atributos da segurança da informação ligados aos dados armazenados nas organizações. Exemplo disso é o papel da criptografia aplicada em ambientes distribuídos, especialmente relacionada à garantia da confidencialidade, fator que apresenta-se como requisito fundamental para a segura utilização dos serviços disponibilizados na rede.

O próximo capítulo apresentará o `OpenAFS` e suas características técnicas, tendo como pano de fundo as ameaças ligadas a ataques de *eavesdropping*, bem como os recursos de segurança que o sistema possui para fazer frente a tais desafios. Desta forma, o `OpenAFS` poderá ser analisado sob a perspectiva dos conceitos apresentados neste capítulo, além de ser comparado a outros Sistemas de Arquivos Distribuídos, com o objetivo de identificar seus pontos críticos e ter mensurados seus níveis de segurança para situá-lo frente às demais soluções apresentadas.

Capítulo 3

OpenAFS

Tal qual introduzido na Seção 2.2, os Sistemas de Arquivos Distribuídos (SAD) foram projetados para disponibilizar, remotamente, arquivos a aplicações através de uma rede, executando tal serviço em um ambiente distribuído. Este capítulo trata de um SAD em particular, denominado `OpenAFS`. As próximas seções apresentam os conceitos gerais a respeito do sistema e listam as características técnicas e de segurança que o sistema disponibiliza nativamente, bem como ressaltam as possibilidades de customizações que podem alavancar os níveis de segurança oferecidos aos dados em armazenamento. Ao final, traça-se um paralelo entre o `OpenAFS` e outros três Sistemas de Arquivos Distribuídos, de modo a justificar sua escolha e estabelecer parâmetros de comparação.

3.1 Conceitos Gerais

A grande diversidade de aplicações atuais são dependentes de ambientes distribuídos que viabilizem o armazenamento de grandes massas de dados e garantam, ainda, alta disponibilidade, integridade e confidencialidade aos dados armazenados.

Exemplos dessa dependência podem ser observados em pequenas, médias e grandes empresas, cujos sistemas necessitam fornecer acesso a arquivos da organização com um mínimo de interrupção possível, além de garantir que tais dados sejam íntegros e acessíveis apenas aos usuários autorizados [48].

Neste sentido se apresenta o ***Andrew Distributed File System (AFS)***, Sistema de Arquivos Distribuídos que fornece uma arquitetura cliente-servidor para compartilhamento de arquivos e replicação de conteúdo distribuído que proporciona independência de local, escalabilidade, segurança e transparência na migração de recursos [16].

No início da década de 1980, a *Carnegie Mellon University* iniciou o Projeto *Andrew*, com o objetivo de fornecer à Universidade um sistema de computação distribuído que conectasse várias estações de trabalho a servidores administrativos. Mais tarde, com seu

amadurecimento, o projeto tornou-se comercialmente disponível, tendo sido adquirido pela IBM em 1998.

Dois anos depois, a IBM decidiu tornar o código publicamente acessível, anunciando então o `OpenAFS` como um *software* de código aberto. O projeto de desenvolvimento permanece ativo, contando com a *The OpenAFS Foundation*[1] para promover a estabilidade e o crescimento deste SAD, mantendo-o como uma implementação *open-source*.

O `OpenAFS` conta com versões para a maioria dos sistemas operacionais modernos, podendo ser citados UNIX, Linux, MacOS X e Microsoft Windows. De acordo com a pesquisa executada por Elomari et al. [49], o `OpenAFS` oferece diversas melhorias em relação aos sistemas tradicionais, tendo como principais características:

- escalabilidade;
- proporciona independência de localização;
- oferece alto nível de segurança a dados em armazenamento e no tráfego de credenciais; e
- provê transparência na migração de recursos.

Utilizado como SAD em grandes universidades ao redor do mundo (por exemplo, a Faculdade de Tecnologia da Informação da Universidade de Stanford[2], o Instituto de Tecnologia de Nova Jersey (NJIT)[3], a Universidade de Tecnologia da Informação de Michigan[4] e a Universidade da Carolina do Norte[5]), o `OpenAFS` é completamente transparente para o usuário, provendo independência de localização e de sistema operacional [19].

Em outras palavras, ele oculta sua natureza distribuída do usuário e, assim, o acesso a arquivos disponibilizados por este SAD ocorre como se fosse um acesso local, com o diferencial para o cliente de poder armazenar um número muito maior de dados remotamente, além de outras vantagens, como a possibilidade de obter cópias de segurança e usufruir de alta disponibilidade.

De acordo com o citado na Seção 2.4.1, um arquivo disponibilizado por um sistema com essas características é acessível a uma aplicação como se fosse local, embora esteja remotamente distribuído. Além disso, como o `OpenAFS` depende do poder das máquinas clientes para o processamento dos dados, o aumento no número de usuários não implica diminuir proporcionalmente o desempenho do sistema, o que confere eficiência e escalabilidade ao ambiente.

[1] http://www.openafsfoundation.org/
[2] https://uit.stanford.edu/
[3] https://www.njit.edu/
[4] https://www.umich.edu
[5] https://www.uncc.edu/

Uma vez traçadas essas perspectivas, as próximas subseções passam a detalhar as peculiaridades técnicas do `OpenAFS`.

3.1.1 Arquitetura

De acordo com Tanenbaum and Steen [48], a arquitetura de Sistemas Distribuídos é um modelo conceitual relacionado principalmente à organização dos componentes do *software* que constituem o sistema. Este desenho revela como os vários componentes são organizados e aponta como deve ser a interação entre eles.

O `OpenAFS` utiliza a arquitetura **cliente-servidor**, com funções específicas para cada ente. Este modelo é largamente utilizado em computação e apresenta dois tipos de máquinas: os servidores, que armazenam dados e executam serviços, e os clientes, que executam aplicações para usuários e acessam dados e serviços fornecidos pelos servidores.

O componente cliente é operado pelo *deamon*[6] `afsd`, responsável pelo suporte ao *cache* local e às operações de acesso aos servidores. O compontente servidor, por sua vez, é dividido em duas categorias: servidores de banco de dados e servidores de arquivos. Cada uma destas categorias são subdivididas em serviços específicos, cujos respectivos processos são ilustrados na Figura 3.1:

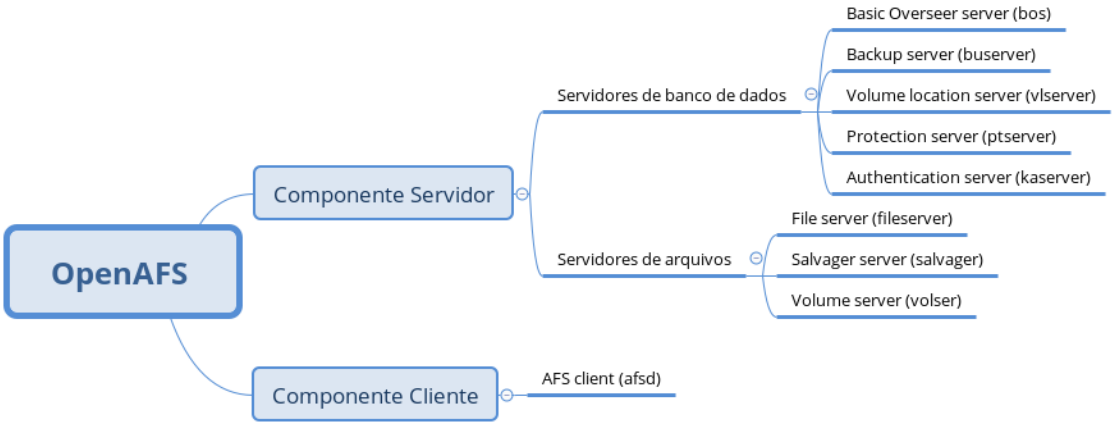

Figura 3.1: Arquitetura do OpenAFS.

Desta forma, os servidores `OpenAFS` armazenam arquivos no Sistema de Arquivos Distribuídos, com diferentes processos gerenciando a entrega e o recebimento destes dados. Os clientes, instalados nas máquinas dos usuários, fornecem acesso aos arquivos armazenados nos servidores e executam um gerenciador de *cache*, o qual permite a comunicação

[6]Um *deamon* é um programa executado como um processo em segundo plano, sob controle do Sistema Operacional.

entre os processos cliente e servidor e atua no controle desta área de armazenamento temporária.

Cabe, neste ponto, detalhar a função de cada dos componentes servidores:

– Servidor de gerenciamento:

  - *Basic OverSeer Server (BOS)*: monitora o status de todos os processos do `OpenAFS`, reiniciando-os automaticamente em caso de falhas;

– Servidores de banco de dados:

  - *Backup Server*: mantém uma base de dados que possibilita a administração de todas as operações de *backup* nos servidores de banco de dados;

  - *Volume Location Server*: mantém um banco de dados com a localização dos volumes, rastreando suas informações de localização, *status*, ID e modificações;

  - *Protection Server*: mantém e manipula informações no banco de dados de proteção, mapeando usuários, grupos e seus identificadores internos; e

  - *Authentication Server*: mantém o banco de dados de autenticação, usado para autenticar usuários e responder a clientes com *tickets* de autenticação.

– Servidores de Arquivos:

  - *File Server*: manipula arquivos e diretórios armazenados nos volumes, provendo a solicitações devidamente autenticadas;

  - *Salvager Server*: executa tarefas de recuperação de dados armazenados nos volumes de um servidor de arquivos; e

  - *Volume Server*: executa tarefas administrativas relacionadas aos volumes, tais como criação, realocação, exclusão e replicação, dentre outras.

Em um servidor tradicional, os discos de armazenamento são subdivididos em partições. Diferentemente disso, o `OpenAFS` trabalha com o conceito de **volumes**. Volumes são subdivisões de partições, e cada um abriga uma sub-árvore de arquivos e diretórios [5]. Em outras palavras, um volume é uma unidade conceitual que denomina um conjunto de arquivos relacionados, armazenados juntos em uma partição de um servidor de arquivos.

O volume é a principal unidade administrativa no `OpenAFS` e possui várias características que facilitam as tarefas administrativas e ajudam a melhorar o desempenho geral do sistema [5], cabendo ser citadas:

– o tamanho relativamente pequeno dos volumes facilita a movimentação entre partições ou entre servidores;

– cada volume corresponde logicamente a um diretório na árvore de arquivos e mantém juntos, em uma única partição, todos os dados que compõem os arquivos no diretório (incluindo subdiretórios possíveis); e

– ao permitirem sua replicação, os volumes aumentam a disponibilidade do sistema.

Estes volumes podem ser movidos de um servidor `OpenAFS` para outro sem que o usuário perceba, já que o sistema os rastreia automaticamente. Tal modelo fornece acesso transparente aos arquivos, de modo que os usuários não necessitam saber em que servidor o arquivo está para ser acessado [19].

Para acessar os arquivos, os clientes consultam um conjunto de servidores administrativos que armazenam em bancos de dados todas as informações relacionadas a volumes e permissões de usuários. Essas máquinas de controle executam *daemons* que localizam o volume ao qual um arquivo pertence, identificam a máquina em que reside fisicamente o volume e verificam as autorizações do usuário para acesso aos recursos solicitados.

Ao conjunto de servidores de controle dá-se o nome de **célula**, que, portanto, forma o domínio administrativo no `OpenAFS`. A Figura 3.2 [19] apresenta um exemplo desta estrutura:

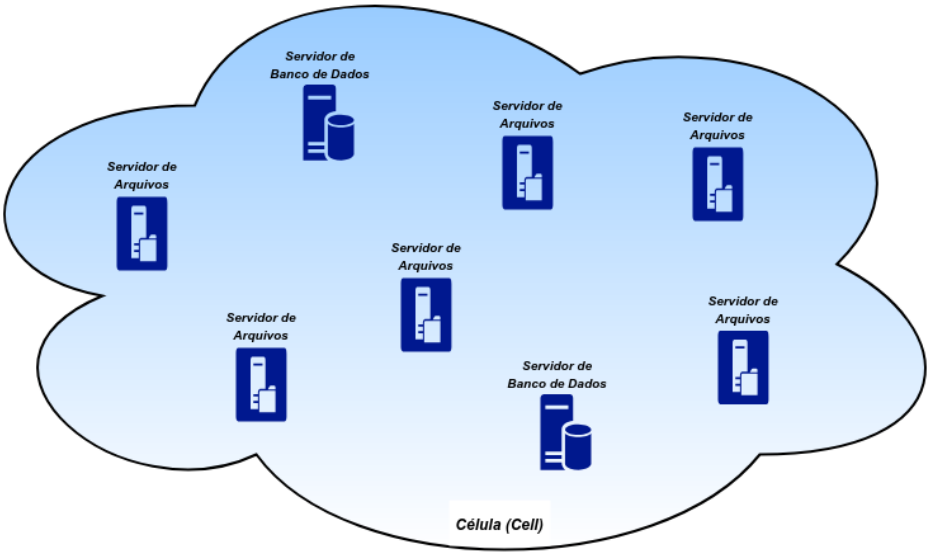

Figura 3.2: Exemplo de Célula do OpenAFS3.

3.1.2 *Cache*

Como citado resumidamente na Seção 2.4.4, *cache* em Sistemas de Arquivos Distribuídos pode ser usado tanto no cliente quanto no servidor. No primeiro caso, o *cache* armazena conteúdo que foi baixado por uma aplicação; no segundo, guarda dados que foram solicitados pela maioria dos usuários da rede.

O `OpenAFS` utiliza o primeiro caso, i.e., cada cliente dedica parte de seus recursos locais (disco e/ou memória) para armazenar dados temporariamente. Essa abordagem é válida, uma vez que a utilização do *cache* no servidor e no cliente, simultaneamente, não tem como consequência o aumento no desempenho do sistema, já que o aumento da taxa de acertos do *cache* no lado do cliente remete a um aumento proporcional de erros no lado do servidor e vice-versa [50].

Nessas circunstâncias, cabe detalhar o mecanismo de controle do *cache* no `OpenAFS`. Cada solicitação de uma dada aplicação cliente a um servidor `OpenAFS` passa, antes de seguir seu destino, por um módulo gerenciador chamado de *"Cache Manager"*. Este módulo é o responsável por se conectar ao respectivo processo no servidor e, recebidos os dados, coordena o armazenamento em *cache*, respondendo, então, à aplicação do cliente com os dados solicitados [5].

A Figura 3.3 ilustra o funcionamento do *cache* no `OpenAFS`:

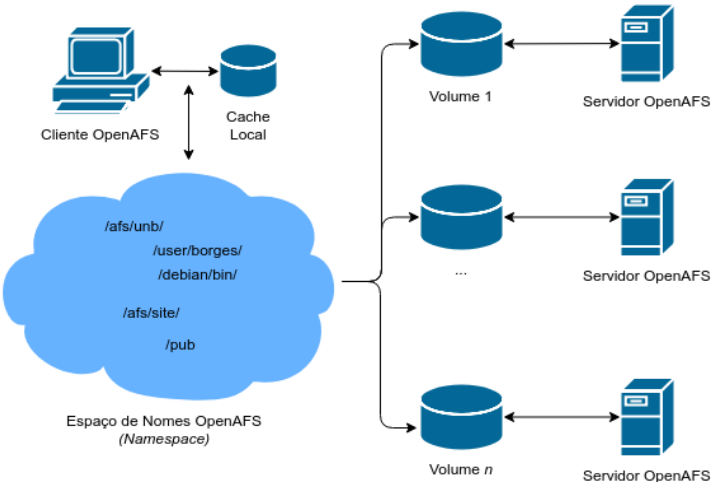

Figura 3.3: Funcionamento do *Cache* no OpenAFS.

Como é possível visualizar na Figura 3.3, os servidores armazenam volumes que, por sua vez, guardam os arquivos e diretórios. Tais conteúdos são acessíveis às estações de trabalho por meio de pontos de montagem no *Namespace*[7] do `OpenAFS`. Então, o

[7]Termo em inglês que pode ser traduzido como "espaço de nomes". Define um delimitador abstrato que fornece um contexto para os itens armazenados.

Gerenciador de Cache, agente que permite o acesso a dados armazenados no espaço de nomes do `OpenAFS`, solicita o arquivo ao servidor apropriado e armazena uma cópia dele no disco local, possibilitando aos programas executados no cliente utilizarem o arquivo em *cache*.

Como vantagem da utilização do *cache* no `OpenAFS` pode ser citado o aumento na velocidade na entrega de dados para as aplicações do cliente, uma vez que, em casos de requisições repetidas a um mesmo arquivo, a aplicação não precisa aguardar o tempo entre sua solicitação e a resposta do servidor. Outro ganho, decorrente deste, é a diminuição de tráfego repetido na rede.

Por outro lado, a consequência indesejada da utilização do *cache* no sistema é a necessidade de se garantir a consistência entre as cópias em *cache* de um dado arquivo e sua versão original. Para fazer frente a este inconveniente, quando um arquivo é modificado no servidor `OpenAFS`, um processo nesta máquina informa a todos os *"Cache Managers"* que possuem cópias daquele dado que aquela versão em cache não é a mais recente.

Este procedimento utiliza um mecanismo especial para realizar tais notificações de forma eficiente, chamado de *"callback"*, algoritmo que age junto ao *Cache Manager*, forçando-o a sempre solicitar a versão mais atualizada de um arquivo [51].

Resumidamente, uma *"callback"* é um algoritmo, normalmente denominado como *cache police* (política de cache), e pode ser entendida como uma "promessa" feita por um servidor a um *Cache Manager*, na qual compromete-se a avisar se um dado anteriormente entregue foi modificado. Com isso, o cliente não precisa contactar o servidor para descobrir se um arquivo armazenado em seu *cache* ainda é válido. Em vez disso, ele pressupõe que o arquivo é válido até que o servidor o informe de modificações [52].

Bžoch and Šafařínk [33] apresentam a importância deste recurso e apontam estes algoritmos como "políticas de substituição", já que eles podem trabalhar com informações estatísticas obtidas a partir de acessos de usuários anteriores para tomar uma eventual decisão de substituição do *cache* por um dado mais novo. Utilizando-se de um simulador de *cache*, os autores efetuaram comparações entre diferentes políticas de *cache* para diversos tamanhos de *cache* em diferentes Sistemas de Arquivos Distribuídos.

Os testes demonstraram, na análise apresentada, que no domínio de uma célula[8] do `OpenAFS`, o algoritmo LFU-SS[9] destacou-se como a melhor política de armazenamento em *cache*, tendo alcançado 4% de melhoria em *caches* de tamanhos pequenos (de até 16MB) e 2% de melhoria em *caches* de tamanhos grandes (maiores ou iguais a 512MB), em comparação com as demais políticas estudadas.

[8] Domínio administrativo do OpenAFS, conforme citado na Subseção 3.1.1.

[9] *The Least Frequently Used with Server Statistics* - este algoritmo substitui as unidades de dados com base na frequência de acessos à unidade e armazena um contador para cada unidade de dados em cache, que é incrementado a cada acesso.

Além da utilização de *"callbacks"*, o Gerenciador de Cache possui um mecanismo chamado *"flushmount"* para rastrear outros tipos de alterações, como a mudança na localização de um volume. Um exemplo deste caso é a situação em que um volume seja movido para outro servidor e o Gerenciador de Cache não tiver acessado nenhum dado durante algum tempo.

Em consequência, o registro de localização de volume poderá estar errado. O recurso de checagem de volumes do Gerenciador de Cache recria a tabela de mapeamentos de nomes e volumes, referenciando volumes recém-realocados [51].

3.1.3 Replicação

A confiabilidade dos Sistemas de Arquivos Distribuídos é baseada, principalmente, nos conceitos de replicação e distribuição [19]. Conforme introduzido na Subseção 3.4, replicar significa criar uma cópia idêntica[10] de um volume em mais de um servidor.

Uma consequência natural da utilização deste recurso é o aumento da disponibilidade, uma vez que a falha de um servidor que hospeda o volume não causa a interrupção no trabalho dos usuários, já que que seu conteúdo permanece disponível em outras máquinas. Para além disso, a replicação fornece balanceamento de carga, uma vez que um servidor não necessariamente será sobrecarregado com acessos a arquivos de um volume com um grande número de requisições [5].

A Subseção 3.1.1 tratou de volumes e suas características. Como preliminarmente citado, o `OpenAFS` permite a execução de clones de volumes, disponibilizando-os em outro(s) servidor(es) por meio de uma interface denominada `Volume Server`.

O `Volume Server` manipula dados do `OpenAFS` no nível de volumes completos, em vez de arquivos e diretórios, possibilitando criar, excluir, mover e replicar volumes. Denominadas como *"replication sites"*, estas cópias são limitadas a um volume habilitado para leitura e gravação, porém permitindo várias réplicas somente-leitura [5].

Por essas razões, cabe concluir que o OpenAFS não se adapta bem a um grande número de leituras paralelas em volumes para leitura e gravação, mas demonstra ser muito bem dimensionado para leituras paralelas de dados em réplicas do tipo somente-leitura. Como a replicação de volumes do tipo somente-leitura não é instantânea, o OpenAFS pode não ser adequado para escrever e ler imediatamente grandes volumes de dados.

[10] Esta cópia é comumente chamada de "clone".

A Figura 3.4 ilustra o processo de replicação no `OpenAFS`:

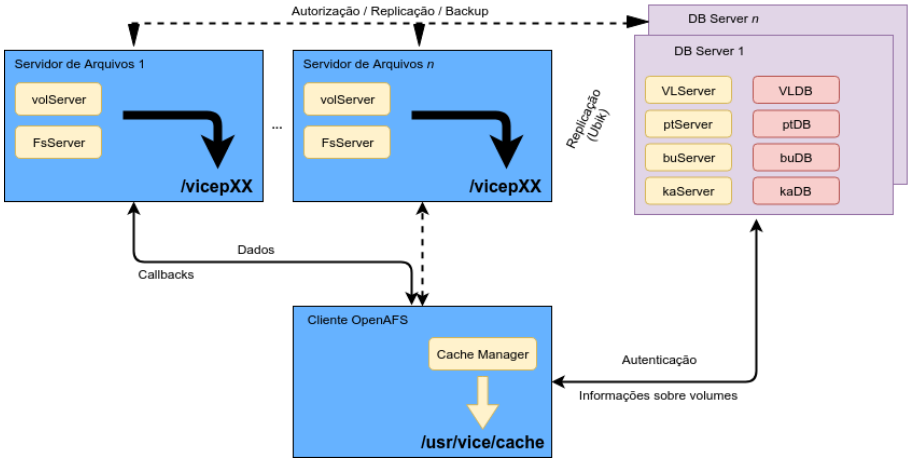

Figura 3.4: Processo de replicação no OpenAFS.

Como é possível observar na figura, o padrão de nomes para as partições que armazenam os volumes no `OpenAFS` é o /vicep*index*, onde *index* é uma letra minúscula. Por convenção, a primeira partição criada é a /vicepa, a segunda /vicepb até /vicepz, podendo naturalmente continuar como /vicepaa, e assim sucessivamente.

O `Ubik` é uma biblioteca de utilitários do `OpenAFS` que mantém as bases de dados do sistema constantemente sincronizadas, vez que as cópias de cada base de dados devem ser sempre idênticas a todo tempo. Trabalhando como componentes cliente e servidor, o processo `Ubik` foi projetado para distribuir as mudanças nas bases de dados do `OpenAFS` de modo automático e instantâneo [5].

Cabe ressaltar, finalmente, que a replicação de volumes em uma célula, disponibilizada nativamente pelo `OpenAFS`, fornece ainda uma maneira alternativa para criação de *backups*, assunto que será apresentado em detalhes na Subseção 3.2.3.

3.2 Segurança

Esta seção trata das principais características de segurança do `OpenAFS` e especifica seus recursos de modo a contextualizar o desenvolvimento dos testes e análises executados no próximo capítulo.

Garantir a segurança em um ambiente de rede é particularmente difícil. Quase todos os procedimentos exigem a transmissão de informações através de estruturas acessíveis a todos os clientes, além do fato de que usuários maliciosos ou não autorizados podem monitorar transações ou até mesmo interceptar as transmissões de dados.

Como citado na Subseção 2.5.2, Sistemas de Arquivos Distribuídos possuem, tal como outros sistemas, a necessidade de controles que garantam disponibilidade, integridade e confidencialidade aos dados neles armazenados. O `OpenAFS` incorpora vários recursos que funcionam em conjunto para fornecer níveis adequados de segurança aos dados nele armazenados [5], cabendo citar:

- aplicação de ACL's em diretórios: arquivos armazenados no `OpenAFS` são protegidos por ACL's associadas ao respectivo diretório, definindo quais usuários ou grupos podem acessar os dados no diretório e controlando a forma com que o acesso é realizado;
- autenticação mútua entre cliente e servidor e entre processos servidores: para estabelecer a comunicação, cada ente exige que o outro prove sua identidade por meio de um processo de autenticação mútua que envolve troca de informações criptografadas, de tal sorte que somente partes válidas podem decifrar e responder;
- *tokens*: para acessar os arquivos do `OpenAFS`, usuários devem provar suas identidades a um serviço de autenticação, fornecendo uma senha. Se este dado for correto, o usuário receberá um *token* como prova de autenticação;
- verificação de autorização: além da autenticação mútua entre cliente e servidor, o `OpenAFS` verifica, ainda, se o cliente cuja identidade foi verificada também tem autorização para fazer a solicitação do acesso, uma vez que pedidos diferentes requerem diferentes tipos de privilégios; e
- criptografia do tráfego de rede: os processos do servidor criptografam informações confidenciais antes de enviá-las aos clientes, impedindo que uma parte não autorizada, mesmo conseguindo capturar o tráfego de uma conexão autenticada, decifre dados criptografados, já que não possui a chave correta.

Cabe citar, neste ponto, que o `OpenAFS` disponibiliza em seu *site* uma página[11] dedicada a documentar o histórico de alertas de segurança emitidos pelo projeto. São divulgadas as vulnerabilidades conhecidas, componentes afetados e uma visão geral que inclui resumo do problema, *link* para o texto completo do comunicado e, quando disponíveis, os *patches*[12] de correção.

As próximas subseções passam a tratar de características específicas de segurança de cada um dos principais mecanismos do `OpenAFS`.

[11] https://www.openafs.org/pages/security/

[12] Termo utilizado para definir programas criados com a finalidade de corrigir um *software*. No caso em tela, o termo *"bugfix"* também pode ser usado, já que se presta a corrigir vulnerabilidades de segurança.

3.2.1 Autenticação

De acordo com Milicchio and Gehrke [19], a confiança é uma questão importante na segurança e, com isso, parte dos problemas nesta área está relacionada à autenticação. Preliminarmente cabe ressaltar que, de acordo com os autores, autenticação é diferente de autorização e, embora estes dois termos estejam sempre ligados, possuem diferentes significados e propósitos. Autenticação tem o propósito de estabelecer a identidade de um ente; autorização, por outro lado, é a decisão de permitir a um ente uma determinada ação. Decorre, deste entendimento, que a autorização é subordinada à autenticação.

Sistemas computacionais utilizam senhas para que um usuário possa provar sua identidade; como consequência, qualquer outro ente que conheça esta credencial pode efetivamente se passar por aquele, causando quebra na confidencialidade. Desta forma, há que se aplicar controles que ofereçam não somente o serviço de autenticação, mas principalmente segurança a este processo, de forma a impedir o roubo ou espionagem de credenciais de acesso.

O `OpenAFS` fornece segurança no processo de autenticação, exigindo que servidores e clientes comprovem suas identidades uns aos outros, antes de trocar informações. Chamado de **autenticação mútua**, o processo exige que o servidor e o cliente demonstrem conhecimento de um "segredo compartilhado" conhecido apenas pelos dois.

A autenticação mútua garante que os servidores forneçam informações somente aos clientes autorizados e que os clientes, por sua vez, recebam informações apenas de servidores legítimos [5].

O `OpenAFS` utiliza dois tipos de autenticação mútua:

- **autenticação mútua simples**: envolve apenas uma chave de criptografia entre cliente e servidor. O cliente se conecta ao servidor enviando uma mensagem criptografada de desafio com uma chave conhecida apenas pelos dois. O servidor decriptografa a mensagem usando sua chave (igual à do cliente) e responde ao desafio usando sua chave para criptografar a resposta. Então, o cliente usa sua chave para decriptografar a resposta do servidor e, se correto, o cliente tem certeza de que o servidor é autêntico. Desta forma, somente alguém que conhece a mesma chave que o cliente pode decriptografar o desafio e respondê-lo corretamente. Pela mesma razão, o servidor pode concluir que o cliente é genuíno. O `OpenAFS` usa este processo para verificar a identidade do usuário durante a primeira parte do procedimento de *login*. Neste caso, a chave de criptografia para a criação dos desafios é a senha do usuário - idealmente conhecida apenas pelo usuário e pelo servidor; e
- **autenticação mútua complexa**: utilizada para todas as outras transações, envolve três chaves de criptografia e três entes - o cliente, o servidor e o *"ticket-*

granter". Antes da comunicação entre um cliente e um servidor, o primeiro entra em contato com um terceiro ente chamado *"ticket-granter"*, autenticando-se mutuamente usando a autenticação mútua simples. Se a transação ocorrer com sucesso, o *"ticket-granter"* fornece ao cliente um *ticket* como prova de que a identidade do cliente foi verificada. O *"ticket-granter"* criptografa o *ticket* com a primeira das três chaves, que é conhecida apenas pelo *"ticket-granter"* e pelo servidor que o cliente deseja contatar. O *"ticket-granter"* envia várias outras informações juntamente com o *ticket*, as quais permitem ao cliente usar o *ticket* efetivamente, apesar de não conseguir decriptografá-lo: uma chave de sessão, que é a segunda chave de criptografia envolvida na autenticação mútua, de modo que o cliente e o servidor possam usar a chave de sessão para criptografar as mensagens enviadas durante as transações; o nome do servidor para o qual o *ticket* é válido; e um limite de tempo de vida do ticket, com duração padrão de 100 horas.

Para executar tais tarefas de autenticação (verificação de identidades para autenticação mútua e geração de *tickets*), o `OpenAFS` utiliza um componente externo ao sistema, chamado `Kerberos`.

O `Kerberos` é um protocolo de autenticação de rede, projetado para fornecer segurança para processos de autenticação usados por aplicativos que utilizam arquitetura cliente-servidor. Usando criptografia simétrica para prova de identidade entre entes em uma rede insegura, o `Kerberos` mantém um banco de dados no qual armazena as chaves de criptografia para usuários (que são suas respectivas senhas) e chave do Servidor `OpenAFS` [53].

Seu serviço de autenticação foi construído em torno do protocolo publicado no final da década de 1970 por Needham and Schroeder [54], tendo sido projetado para fornecer um serviço de autenticação segura e distribuída, utilizando-se de criptografia de chave simétrica. Com relação à criptografia aplicada ao `Kerberos 5`, (versão mais recente), cabe ressaltar o suporte à cifra `Advanced Encryption Standard (AES)` de bloco de 128 bits e tamanhos de chave de 128 ou 256 bits, com a função `SHA-1` como função de *checksum*, conforme descrito em [55].

O papel deste protocolo é fundamental para a garantia da segurança no processo de autenticação não somente dos usuários que desejam se autenticar no servidor `OpenAFS`, mas também na confidencialidade dos dados que trafegam entre os servidores da célula `OpenAFS`.

Cabe ressaltar, em complemento ao apresentado no início desta subseção, que a autenticação é o primeiro passo no processo de autorização. A posse de um *ticket* fornecido pelo `Kerberos` confere ao cliente tão somente a prova de que ele é quem alega ser. A autorização é o próximo passo, cujos procedimentos determinam se o cliente pode ou não

usar um determinado serviço e quais permissões e tipos de acesso ele possui. Este tema será abordado em profundidade na próxima subseção.

É importante definir os principais conceitos e entes envolvidos no processo de autenticação do `Kerberos`:

- *Realm*: Conjunto de nós (estações de trabalho, servidores, etc.) que compartilham o banco de dados `Kerberos`;
- *Principal*: Identidade de um cliente, usuário ou servidor, à qual os *tickets* do `Kerberos` podem ser atribuídos;
- Cliente: Processo, *host*, servidor ou outro nó que faz uso do `Kerberos` sob o comando de um usuário;
- Servidor: Principal em particular, que fornece um recurso para clientes da rede;
- Serviço: Recurso fornecido pelo Servidor (ou por um conjunto destes) aos clientes da rede;
- *Key Distribution Center (KDC)*: Serviço do `Kerberos`, composto pelo *Authentication Server (AS)* e pelo *Ticket Granting Server (TGS)*. Atende tanto às solicitações de *ticket* inicial quanto de permissão de concessão de *tickets*;
- *Authentication Server (AS)*: Componente do KDC que manipula a solicitação inicial e emite um TGT. É o responsável por manter o banco de dados de entidades e as respectivas chaves secretas;
- *Ticket Granting Server (TGS)*: Componente KDC que manipula a etapa de concessão de *tickets* para os serviços solicitados.
- TGT: *Ticket* destinado ao TGS que pode ser usado para obter *tickets* para serviços adicionais, sem a necessidade de senha ou de chave secreta do usuário original;
- *Service Ticket*: *Ticket* emitido pelo TGS que pode ser usado para autenticar um serviço;

A Figura 3.5 apresenta um esquema lógico da sequência de ações no processo de autenticação:

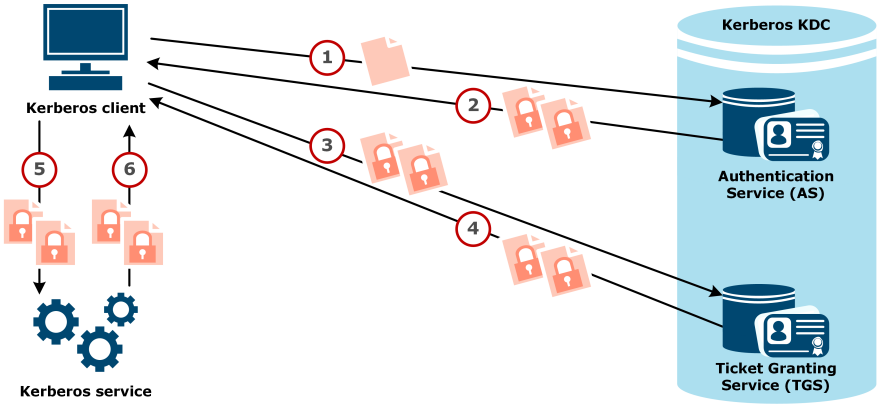

Figura 3.5: Funcionamento geral do Kerberos [1].

Uma vez que o processo de autenticação do `OpenAFS` será analisado em detalhes no Capítulo 4, cabe identificar, neste ponto, a interação deste SAD com o `Kerberos`. Observando-se a Figura 3.5 é possível identificar três etapas principais:

- os passos 1 e 2 formam a etapa de autenticação inicial, na qual o cliente solicita um *token* de autenticação e o recebe;
- os passos 3 e 4 formam a etapa de solicitação de serviço, na qual o cliente solicita a concessão de um *ticket* para acesso a um determinado serviço (`OpenAFS`) e o recebe; e
- os passos 5 e 6 formam a etapa de acesso ao serviço desejado (`OpenAFS`), na qual o cliente usa o *ticket* para acessá-lo.

Ainda de acordo com a Figura 3.5, é possível identificar que o *Authentication Server (AS)* e o *Ticket Granting Server (TGS)* formam o conjunto principal do `Kerberos`, denominado *Key Distribution Center (KDC)*. Além destes mecanismos, existe a base de dados do KDC, responsável por armazenar todas as chaves secretas, sejam de serviços ou de usuários, além de dados intrínsecos das contas armazenadas, tais como data de criação, políticas de acesso, dentre outros.

Dada a relevância do processo de autenticação apresentado nesta Subseção, justifica-se a apresentação de uma análise a respeito deste serviço, conforme objetivos específicos desta dissertação, citados na Subseção 1.3.2. Para isso, a avaliação técnica da autenticação do `OpenAFS` no `Kerberos` será discutida no decorrer do próximo capítulo, acompanhada de testes e seus respectivos resultados.

É importante salientar, por oportuno, que o `Kerberos` versão 5 apresentou problemas na aplicação de criptografia para proteção de credenciais em conjunto com o `OpenAFS`,

evidenciando vulnerabilidade a ataques de força bruta contra o algoritmo utilizado para a proteção das credenciais de acesso.

Outro protocolo utilizado especificamente na autenticação em servidores `OpenAFS` é o Extended Remote Procedure Call (RX). Transportado pelo protocolo UDP, o `RX` é um protocolo de Remote Procedure Call (RPC) que fornece suporte geral para transferência de dados, autenticação e segurança de ponta a ponta.

Apesar de não ser definido por RFC[13], o `RX` é robusto o suficiente para permitir que clientes e servidores se adaptem às diferenças em seu desempenho relativo, a links com perdas e a atrasos da rede, fornecendo, em baixo nível, a garantia de entrega de pacotes a um servidor apto a recebê-los [56].

Uma chamada típica entre cliente e servidor `OpenAFS` consiste em:

- O cliente envia uma requisição ao servidor, contendo a ID do serviço;
- O servidor recebe a requisição e envia um desafio ao cliente;
- O cliente responde ao desafio, enviando o *token* de acesso (recebido anteriormente do `Kerberos`);
- O servidor decifra o *token* e envia uma mensagem de resposta autorizando o acesso do cliente; e
- O cliente confirma o recebimento com uma mensagem do tipo `ACK`.

No `RX`, todas as conexões autenticadas são, potencialmente, seguras. Com isso, conexões usando o `RX` podem ser protegidas contra alteração de dados usando criptografia, normalmente com uso de um segredo compartilhado entre cliente e servidor. Este protocolo é usado nas fases 5 e 6, ilustradas na Figura 3.5, e será objeto de análise na Subseção 4.2.3

3.2.2 Controle de Acesso

O controle de acesso a arquivos no `OpenAFS` é executado por um mecanismo chamado ACL *(Access Control List)*. As listas de controle de acesso são relações de permissões ligadas a um determinado objeto.

No contexto do `OpenAFS`, elas definem quais usuários e grupos podem acessar dados em determinado diretório, e de que forma podem fazê-lo, ou seja, determinam quem pode ver, alterar ou mover os arquivos. Em outras palavras, as ACL mapeiam usuários e/ou grupos a um conjunto de direitos de acesso.

[13]Publicações que documentam os padrões, serviços e protocolos oficiais e não-oficiais da Internet.

Antes de tratar especificamente das ACL, faz-se necessário assinalar as diferenças no controle de acesso a dados implementado pelo sistema de arquivos do `OpenAFS` e o sistema de arquivos UNIX, vez que ambos se comportam de modo semelhante, mas com detalhes distintos.

O UNIX controla ao nível de arquivo associando 9 bits para cada arquivo (leitura, escrita e execução para donos, grupos e outros usuários); diferentemente disso, o `OpenAFS` provê controle de acesso pelas ACL ao nível de diretório, protegendo todos os arquivos em um mesmo diretório da mesma maneira [5]. Desta forma:

- mover um arquivo para outro diretório causa alteração nas permissões de acesso a ele aplicadas, que passam a ser as da ACL do destino;
- a alteração de uma ACL provoca alteração na proteção de todos os arquivos do diretório por ela controlado; e
- criar uma subpasta faz com que ela herde automaticamente as permissões de sua pasta pai e implica cópia da ACL do diretório pai, mas a ACL filha pode ser alterada independentemente da ACL pai, desde que esta conceda tal acesso a um usuário.

Em células (Subseção 3.1.1) nas quais o compartilhamento de arquivos é amplamente difundido, pode não ser desejável que todos os usuários tenham o mesmo tipo de acesso a todos os arquivos, e controles devem ser implementados, conforme Bžoch and Šafařík [24].

O `OpenAFS` define sete direitos possíveis para configuração de controle de acesso a um diretório e aos arquivos que ele contém: **a** - *administer*; **d** - *delete*; **i** - *insert*; **k** - *lock*; **l** - *lookup*; **r** - *read*; e **w** - *write*.

3.2.3 *Backup*

Como foi comentado na Seção 2.5.2, mídias mecânicas sofrem panes naturais ao final de seu ciclo de vida, dada a natureza da estrutura de seus componentes. Para contornar este problema, mesmo em sistemas que possuem recursos de replicação de dados, *backups* devem ser detalhadamente planejados, com a finalidade de diminuir a probabilidade de perda de dados em caso de falhas de *hardware*.

Cópias de segurança podem ser realizadas nativamente pelo `OpenAFS` com a configuração de um processo opcional denominado *`Backup Server`*, que atua em conjunto com outro serviço chamado *`Backup Database`*.

O *`Backup Database`* é uma base de dados administrativa, replicada e controlada pelo *`Backup Server`* no contexto de uma célula do `OpenAFS`. Juntos, os serviços possibilitam

o *backup* de volumes inteiros em fita ou mesmo em outro sistema de arquivos, fornecendo a possibilidade de restauração quando necessário [5].

Alternativamente, o `OpenAFS` possibilita a gravação do estado de um volume em um dado momento e, em seguida, o armazena em outras mídias ou em outro sistema de arquivos, de modo a possibilitar a recuperação de arquivos excluídos ou alterados acidentalmente [5].

Neste ponto cabe ressaltar um requisito fundamental para a operação de Sistemas Distribuídos de um modo geral, que afeta não somente a operação de *backup*, mas de toda a tecnologia envolvida nos bancos de dados distribuídos do `OpenAFS`: a imperiosa necessidade de sincronia dos relógios para a operação correta de todos os serviços.

Coulouris et al. [7] ressaltam que cada computador em um sistema distribuído possui seu próprio relógio interno, que pode ser usado por processos locais para obter o horário atual. Desta forma, dois processos em execução em computadores diferentes podem associar cada data e hora a seus eventos. No entanto, mesmo que os dois processos leiam seus relógios ao mesmo tempo, seus relógios locais podem fornecer valores de tempo diferentes. Latha and Shashidhara [57] completam este raciocínio afirmando que, em um ambiente distribuído, os relógios de cada um dos nós devem concordar com um valor de tempo comum, necessitando de uma sincronia para limitar os erros decorrentes de diferenças de velocidades de *clock* que se acumulam no tempo.

Não somente os Sistemas Distribuídos em si dependem da sincronia da hora entre seus clientes e servidores. No caso do `OpenAFS`, o sistema de autenticação utilizado, provido pelo `Kerberos`, tolera uma variação mínima entre os relógios do servidor e dos clientes, de modo que a sincronização de tempo entre os *hosts* do ambiente de testes deve ser adequadamente implementada. A fim de satisfazer tal necessidade, este trabalho prevê a utilização do `OpenNTPD`[14], que implementa o protocolo NTP, conforme citado na Seção 4.3.

Voltando ao sistema de *backup* do `OpenAFS`, é possível verificar que o mesmo apresenta grande granularidade na configuração, permitindo o amplo controle de opções e automação do processo de criação das cópias de segurança, incluindo itens como a frequência, quais volumes devem ser copiados e características dos *dumps*[15] gerados pelo sistema [5].

Uma vez que o processo de *backup* aumenta o nível de disponibilidade da solução, como previamente assentado na Subseção 2.5.1, é cabível analisar tal subsistema do `OpenAFS` sob a ótica do citado atributo de modo a validar sua eficácia. A execução da análise deste recurso, bem como suas decorrências, serão apresentados no próximo Capítulo.

[14] http://www.openntpd.org/

[15] No contexto do `OpenAFS`, é uma coleção de dados resultante do *backup* de um volume ou conjunto de volumes. Semelhante à terminologia tradicional de processos de *backup*, os *dumps* no `OpenAFS` podem ser do tipo completo ou incremental, mas não há geração de *backup* diferencial.

3.2.4 *Accountability*

A definição do termo *Accountability*, explanada na Subseção 2.5.1, denota a necessidade de responsabilização como um dos atributos primários para a segurança em Sistemas de Armazenamento. Por esta razão, um sistema de auditoria deve fornecer meios para registro de informações relacionadas à segurança, bem como ferramentas de alerta sobre potenciais ou reais violações da política de segurança do sistema auditado [58].

O `OpenAFS` não possui, nativamente, um serviço de auditoria de seus eventos. Entretanto, possibilita o envio de informações de auditagem do sistema para outro serviço especializado, que pode organizar e sistematizar o processo.

A documentação do `OpenAFS` sugere a utilização do `AIX Audit` para prover o serviço de auditoria a partir das informações fornecidas pelo sistema [5]. O `AIX Audit` é um subsistema utilitário que fornece uma organização lógica de dados que facilita a identificação de eventos que potencialmente estejam ligados a ameaças à política de segurança do sistema monitorado [59].

Como já citado, o `OpenAFS` fornece informações a respeito de várias classes de eventos [5], tais como:

- autenticação;
- processos de *backup*;
- processos do servidor;
- processos do gerenciador de *cache*;
- uso e tentativas de abuso de privilégios de acesso;
- eventos de criação e apagamento de objetos; e
- modificação de atributos.

Enviando estas informações de *syslog*[16] para um `AIX Audit`, o `OpenAFS` passa a contar com um sistema que possibilita registrar ocorrências de acesso de clientes a objetos e detectar repetidas tentativas de ignorar mecanismos de proteção ou, ainda, identificar uso indevido de privilégios.

3.3 Estudo comparativo

Esta seção apresenta as características gerais, de arquitetura e de segurança de alguns Sistemas de Arquivos Distribuídos de referência existentes, com os objetivos de fornecer

[16]Padrão criado pela IETF para a transmissão de mensagens de *log* em redes IP.

uma visão geral a respeito de cada um e de possibilitar a análise comparativa entre tais SAD e o `OpenAFS`.

3.3.1 NFS - *Network File System*

Conceitos gerais

Desenvolvido pela *Sun Microsystems* no início da década de 1980, o NFS é o Sistema de Arquivos Distribuídos utilizado nativamente pelos sistemas Unix. A abstração permite que um cliente acesse, via rede, um sistema de arquivos remoto como se este fosse local, disponibilizando, desta forma, áreas de armazenamento para compartilhamento de arquivos em rede, de forma hierárquica e unificada [7].

O protocolo NFS é utilizado pelo cliente para montar[17] um sistema de arquivos de determinado servidor, antes de acessar os arquivos nele armazenados. A versão atual, NFSv4, inclui melhorias de desempenho, aplicação de criptografia e, como principal modificação, utiliza um protocolo *stateful*[18] para transporte, diferente do *stateless*[19] usado nas versões anteriores [60].

Cada operação do NFS é projetada para ser idempotente[20]. Assim sendo, na hipótese de travamento do servidor em meio a uma operação, o cliente poderá reexecutá-la sem a necessidade de ter a confirmação de que o procedimento tenha sido finalizado.

A comparação com este Sistema de Arquivos Distribuídos justifica-se porque, nos últimos anos, o NFS tem sido usado em ambientes onde o desempenho é um fator importante, como por exemplo em *clusters* Linux de alto desempenho. A versão mais recente, NFSv4, fornece um recurso denominado pNFS (NFS paralelo), que permite acesso direto de clientes a vários dispositivos de armazenamento, melhorando assim o desempenho e a escalabilidade do sistema de arquivos, conforme Lim and Yang [61].

Arquitetura

A arquitetura do NFS segue o padrão cliente-servidor, sendo este o responsável por exportar os volumes e aquele o encarregado de realizar a interface entre o sistema de arquivos compartilhado e o cliente local.

[17] Termo usado para definir a ação de anexar um determinado sistema de arquivos, remoto ou não, ao dispositivo em utilização.

[18] Serviços que utilizam este tipo de protocolo mantém uma tabela de estados de conexões anteriores, o que garante persistência das requisições. Em casos de pane, apenas os dados perdidos devem ser reenviados.

[19] Protocolos desta natureza não guardam estados de conexões passadas. No cenário de utilização do NFS, significa que clientes que perderem a conexão com um servidor continuarão a enviar novas solicitações, podendo causar saturação da rede.

[20] Operações desta natureza podem ser repetidas várias vezes, sem causar problemas na execução.

A Figura 3.6 ilustra a arquitetura do NFSv4:

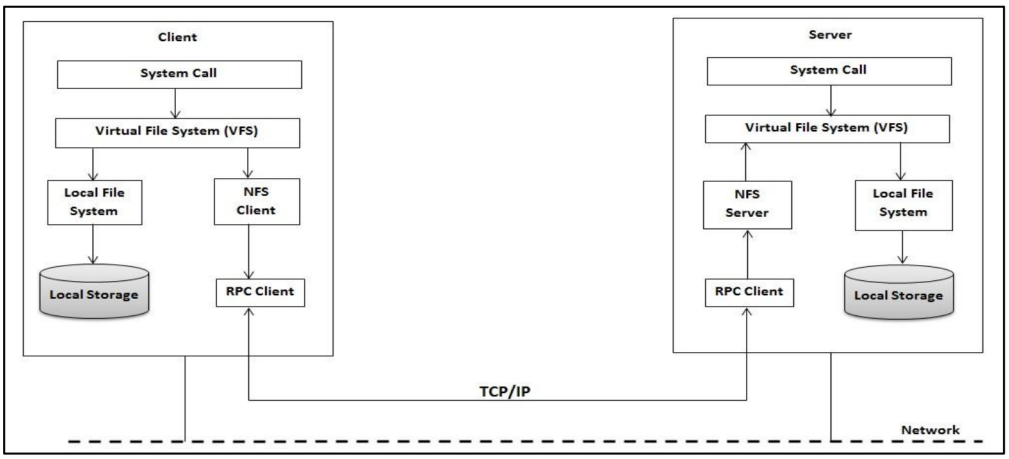

Figura 3.6: Arquitetura básica do NFS [2].

Como é possível observar, o acesso do cliente ao servidor NFS se dá através de uma conexão de rede por meio de RPC. O Virtual File System (VSF) fornece o suporte para vários sistemas de arquivos simultaneamente em um mesmo host, entendendo a solicitação e oferecendo o sistema de arquivos adequado para atender ao pedido.

Segurança

O NFSv4 apresenta uma ruptura com as versões anteriores, apresentando modificações inclusive em questões tidas como alicerces da antiga versão [62], como a utilização mandatória do protocolo TCP em substituição ao UDP.

Foram implementados diversos mecanismos de segurança usando criptografia forte, inclusive suportando negociação do nível de segurança com o cliente. A versão 4 do protocolo continua baseando-se no modelo de segurança do Remote Procedure Call (RPC), mas exige a implementação do esquema RPCSEC_GSS, baseado na *Generic Security Services* API[21] (GSS-API).

Um dos objetivos desta interface é obter isolamento entre a camada de aplicação e os serviços de segurança propriamente ditos, tornando o esquema de segurança transparente para a aplicação [62], conforme é possível observar na Figura 3.7:

[21]Padrões definidos por um dado software que podem ser usados por outros aplicativos que precisem usar suas funcionalidades, mas não necessariamente necessitam manipular os detalhes de sua implementação - somente utilizar seus serviços.

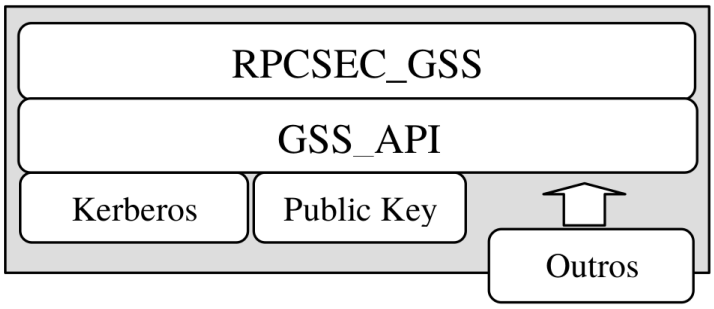

Figura 3.7: Esquema de segurança do NFSv4 [2].

Diferentemente de outros esquemas que só proveem autenticação, o RPCSEC_GSS pode garantir integridade e privacidade do corpo inteiro de uma mensagem RPC [62]. Análises aprofundadas sobre a segurança do NFSv4, proporcionada pela utilização do esquema RPCSEC_GSS, podem ser consultadas em [2] e [61].

3.3.2 GFS - *Google File System*

Conceitos gerais

O Google File System (GFS), também conhecido como GoogleFS, é, tal como sistemas semelhantes, um SAD escalável que armazena grandes quantidades de dados (cada *cluster* com aproximadamente 1000 nós e 300 TB de capacidade de armazenamento) e os disponibiliza para um grande número de aplicações que exigem acesso intensivo, ao mesmo tempo que oferece alta disponibilidade, confiabilidade e performance [3].

Projetado no início dos anos 2000, o GFS é usado para suportar elevados requisitos de processamento de dados em constante expansão, característicos da operação de dados de seu ambiente. Otimizado para operar eficientemente com grandes arquivos, o sistema permite acesso simultâneo a milhares de clientes simultâneos com o mínimo possível de carga nos servidores.

Arquitetura

Um *cluster* GFS é formado por um único servidor *master* e vários *chunkservers*, estes tipicamente executando Linux, com a função de armazenar os arquivos propriamente ditos. Os arquivos são divididos em *chunks*, que são como blocos de um sistema de arquivos tradicional, mas de 64MB de tamanho, ou seja, maiores que um bloco usual.

A Figura 3.8 ilustra a arquitetura do GFS:

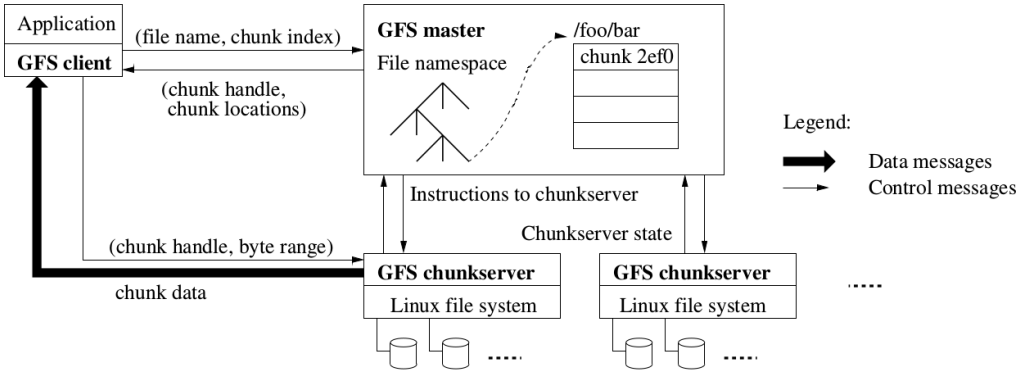

Figura 3.8: Arquitetura do GoogleFS [3].

É possível verificar na figura que o servidor *master* atua como o coordenador do *cluster*, recebendo e enviando mensagens de controle (*logs* de operação), que minimizam interrupções do serviço (já que se o servidor mestre falhar, um servidor substituto que monitorou o log da operação poderá substituí-lo). Outro tipo de mensagem de controle são os metadados, que informam ao servidor mestre a quais arquivos um determinado fragmento pertence e onde ele se encaixa no arquivo.

Com os arquivos armazenados nos *chunkservers*, os metadados (nome do arquivo, respectivo *chunk*, localização de cada *chunkserver* e réplicas) são guardados em memória no *master*, que por sua vez tem seu estado replicado, como forma de *backup*. O armazenamento em memória busca fornecer mais velocidade nas operações.

Sendo o *master* o ponto central em que são armazenadas todas as informações sobre *chunks*, é com ele que as aplicações se conectam inicialmente, para então se conectar ao *chunkserver*, que efetivamente armazena o dado desejado [3].

Segurança

Com relação à confidencialidade, o GFS, por ser um sistema proprietário, não teve publicado oficialmente seu esquema de segurança específico e não há publicações na literatura que abordem esta propriedade de segurança.

No tocante à disponibilidade, o GFS oferece recursos de tolerância à falhas com a execução de replicação de *chunks* em múltiplos *chunkservers*. São armazenadas, por padrão, três réplicas, cujos metadados são armazenados no *master*, conforme citado no item anterior. Da mesma forma, os *logs* de operações são armazenados e replicados, dada sua criticidade.

3.3.3 HDFS - *Hadoop Distributed File System*

Conceitos gerais

Assim como o GFS, o Hadoop Distributed File System (HDFS) apresenta várias similiaridades com os demais SAD. Por outro lado, apresenta significativas diferenças, as quais serão tratadas a partir deste ponto.

Este SAD é uma implementação de código aberto lançado em 2008 que integra o conjunto de tecnologias Hadoop[22] e fornece um sistema escalável para armazenamento de *pools* de *big data*, usados com aplicações específicas de análise *clusterizada* a partir de grandes massas de dados estruturados e não-estruturados.

Baseado no Google File System (GFS), o HDFS também armazena seus arquivos em blocos de 64Mb e, assim como ocorre em outros SAD, o HDFS suporta a tradicional organização hierárquica de arquivos, possibilitando a criação de diretórios e as habituais operações com arquivos.

Arquitetura

Também semelhante ao GFS neste ponto, o HDFS possui uma arquitetura *master* e *slave*. Um *cluster* é composto por um único NameNode, que é o servidor principal que gerencia o sistema de arquivos e controla o acesso dos clientes, e vários DataNodes.

A Figura 3.9 apresenta a arquitetura do HDFS:

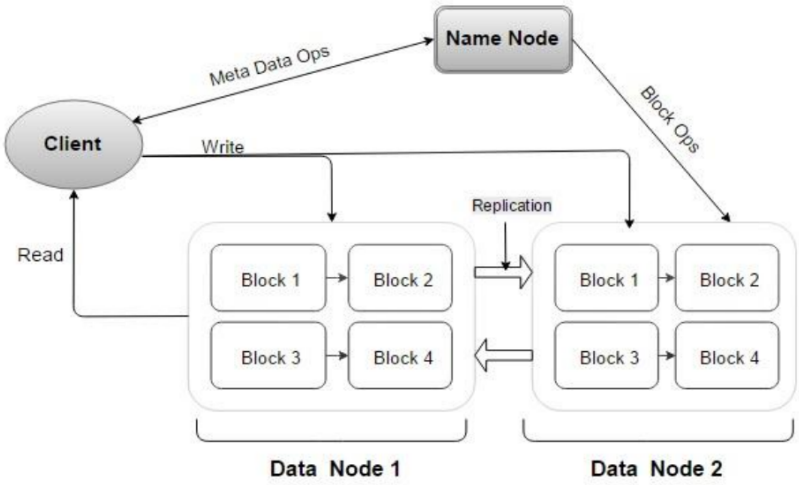

Figura 3.9: Arquitetura do HDFS [4].

Análogo ao GFS, na arquitetura HDFS existe um nó mestre chamado de *NameNode*. Ele armazena os metadados em memória, para rápida recuperação do cliente. Por sua vez,

[22]http://hadoop.apache.org/

o *DataNode* armazena os dados reais no sistema de arquivos, efetuando as operações de leitura e escrita, conforme a solicitação do cliente, e o conjunto de *DataNodes* possibilita a replicação dos dados, provendo a tolerância à falhas da solução.

Segurança

De acordo com Cohen and Acharya [63], dados confidenciais armazenados em uma infraestrutura HDFS são potenciais alvos para exfiltração[23], corrupção, acesso não autorizado e modificação. Nas palavras dos autores, o modelo de segurança do Hadoop toma como pressupostoso algumas suposições que normalmente não são reais nas redes das organizações, tais como acesso de *root* a usuários das máquinas do *cluster* ou possíveis modificações em pacotes da rede.

Neste mesmo sentido, um dos principais colaboradores do projeto de segurança do Hadoop, Owen O'Malley, citou em seu *site* [64] que "a motivação para adicionar segurança ao Apache Hadoop na verdade tinha pouco a ver com as noções tradicionais de segurança na defesa contra *hackers*, já que todos os grandes clusters do Hadoop estão protegidos por *firewalls* corporativos que só permitem o acesso de funcionários". Desta forma, como já introduzido, um invasor que tenha como alvo específico este ambiente, considerando as oportunidades certas, pode obter acesso não autorizado ao HDFS [63].

Com relação à disponibilidade, o Hadoop Distributed File System (HDFS) foi projetado para armazenar de maneira confiável arquivos muito grandes em máquinas em um grande *cluster*. Ele armazena cada arquivo como uma seqüência de blocos, que são por sua vez replicados para tolerar falhas. O tamanho do bloco e o fator de replicação são configuráveis por arquivo [65].

3.3.4 Tabela comparativa

Esta subseção apresenta uma comparação onde foram dispostos os dados do `OpenAFS`, do NFS, do GFS e do HDFS.

O estudo de Adamov [4] apresenta uma comparação entre o HDFS e o GFS, apontando as diferenças a respeito de diversos critérios. Já Milicchio and Gehrke [19] procederam a uma comparação entre o `OpenAFS` e o NFS sob vários outros aspectos.

Tais estudos subsidiaram a comparação efetuada a seguir, ilustrada na Tabela 3.1:

[23] Terminologia recente, ligada à Segurança da Informação, que define a transferência não autorizada de dados armazenados em um dado sistema.

Tabela 3.1: Tabela comparativa.

	Disponibilidade	**Integridade**	**Confidencialidade**	**Cache**
OpenAFS	Por replicação	GSS-API	Kerberos	No cliente
NFS	Sem replicação	GSS-API	Kerberos	Por delegação
GFS	Por replicação	Checksums	Não publicada	No cliente
HDFS	Por replicação	Checksums	AES	No cluster

Da análise da tabela e da leitura dos tópicos apresentados, é possível verificar que para uma organização de médio porte que não apresente requisitos de análise de *big data*, necessitando tão somente de um sistema de compartilhamento de arquivos que ofereça condições de escalabilidade, o `OpenAFS` apresenta-se como solução viável, o que justifica a presente análise de segurança.

3.4 Considerações finais

Este capítulo apresentou o `OpenAFS`, evidenciando o detalhamento das características técnicas e particularidades do sistema, além de tratadas suas possibilidades e limitações. Especial atenção foi dada à parte dedicada à segurança do sistema, em que foram discutidas as funções de autenticação, controle de acesso, *backup* e responsabilização. Tratar das especificidades de segurança do `OpenAFS` oferece as condições teóricas necessárias para o entendimento dos testes realizados a seguir.

Ao final, foi apresentado um sucinto estudo comparativo entre o `OpenAFS` e outros três SAD, de modo a identificar as diferenças entre eles e complementar a abordagem do presente estudo.

O capítulo que se segue fornecerá os resultados obtidos com os testes de segurança realizados com a simulação de operação de uma célula do `OpenAFS`.

Capítulo 4

Análise de segurança

Este capítulo apresenta a execução dos testes realizados com a finalidade de aferir a capacidade dos subsistemas do `OpenAFS` relacionadas à segurança, conforme planejado no decorrer dos Capítulos anteriores.

A análise de uma implementação do `OpenAFS` em um ambiente simulado é a parte final dos passos delineados na Subseção 1.4.3. De acordo com Gerhardt and Silveira [22], citados na introdução desta dissertação, a execução desta etapa é necessária para testar a hipótese proposta na Subseção 1.4.2.

A montagem do ambiente de testes utilizou a virtualização como ferramenta para simular a operação de uma máquina na função de cliente, bem como de uma célula típica do `OpenAFS` e da infraestrutura de autenticação do `Kerberos`. Este cenário compõe a arquitetura básica deste SAD, o que possibilita o monitoramento dos recursos dos entes envolvidos e a coleta de tráfego, necessário para a análise planejada.

Tal tráfego, depois de coletado, passará por uma análise circunstanciada, comparando-se os resultados obtidos com as características apresentadas na Subseção 3.2. A próxima Seção apresenta a arquitetura do ambiente simulado.

4.1 Arquitetura

Como foi tratado na Subseção 3.1.1, a arquitetura do `OpenAFS` é do tipo cliente-servidor, típica de Sistemas de Arquivos Distribuídos, sendo baseada em uma estrutura específica denominada célula.

Uma célula do `OpenAFS` é definida como sendo um conjunto de servidores e clientes gerenciados e operados por uma organização administrativamente independente. Em uma célula, ao menos uma das máquinas deve executar os processos do servidor de banco de dados e do servidor de arquivos.

Sendo esta a estrutura básica do OpenAFS, justifica-se utilizá-la como base para a realização dos diferentes testes propostos neste trabalho. Para tanto, foi implementado um cenário virtualizado do OpenAFS que possibilitou a ocorrência de tráfego de dados específico do sistema, conforme se observa na Figura 4.1, que ilustra sua arquitetura:

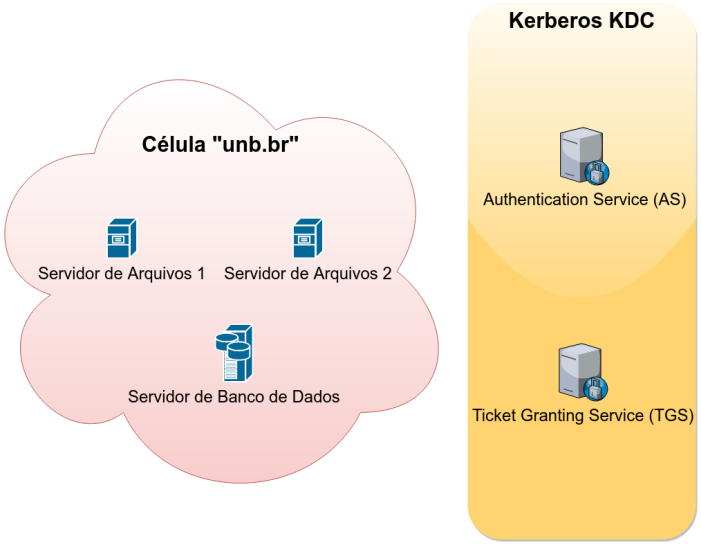

Figura 4.1: Arquitetura do cenário.

Como já comentado, o objetivo desta implementação é possibilitar a obtenção de tráfegos de rede para observação técnica. A análise de tais fluxos de dados possibilitará visualizar e mapear as características de segurança que o sistema oferece aos dados que gerencia. Esta análise faz parte dos objetivos desta dissertação e é parte crucial para a identificação das propriedades de segurança da informação do OpenAFS.

Cabe ressaltar, também, que esta proposta considera a presença de pelo menos um intruso passivo que tem capacidade de monitorar qualquer quantidade de nós do sistema, com os objetivos tanto de colher informações sensíveis (credenciais, por exemplo) quanto de obter dados armazenados por meio da remontagem de tráfego do OpenAFS durante a execução dos processos de replicação de volumes[1].

4.1.1 Configurações

A apresentação dos detalhes da instalação e configuração geral do ambiente proposto para análise foge ao escopo deste trabalho. Todavia, é importante evidenciar que a montagem do cenário seguiu a documentação oficial do Projeto OpenAFS e do Projeto MIT Kerberos,

[1]Conforme detalhado na Subseção 3.1.3.

especialmente o *"OpenAFS Administration Guide"*[2], o *"OpenAFS Quick Start Guide for UNIX"*[3] e o *"The MIT Kerberos Administrator's How-to Guide"*[4].

De maneira resumida, a montagem do ambiente foi realizada da seguinte forma:

- Criação da máquina virtual[5] `"kerb"`, com Sistema Operacional Debian GNU/Linux 9.7 (stretch) que executa, dentre outros, dois serviços críticos para o funcionamento do ambiente: a) o `Kerberos` 5 KDC (*Key Distribution Center*), que verifica a identidade dos usuários e concede *tickets* que, em seguida, são convertidos em *tokens* para provar ao servidor a autenticidade do usuário; e b) O `OpenNTPD`, que implementa o protocolo NTP, responsável por sincronizar os relógios internos das máquinas da rede com o seu próprio, possibilitando a uniformidade dos *clocks* e permitindo o correto funcionamento da tecnologia de banco de dados distribuído do `OpenAFS`;
- Criação da VM "`afs1`", com Sistema Operacional Debian GNU/Linux 9.7 (stretch) que executa o componente Servidor do `OpenAFS` versão 1.6.20 e todos os seus processos, tais como descritos na Subseção 3.1.1;
- Criação da VM "`afs2`", com as mesmas características da VM "`afs1`" e que, em conjunto com ela, formam o domínio principal do `OpenAFS`, denominado `célula`, neste trabalho configurada como "`unb.br`"; e
- Criação da VM "`client`", com Sistema Operacional Debian GNU/Linux 9.7 (stretch) que executa o componente Cliente do `OpenAFS`, usada para acessar os recursos do Servidor `OpenAFS` após autenticação no KDC.

Exposta a estrutura básica do ambiente de testes, são apresentados, a partir deste ponto, os detalhes de configuração afetos às características de segurança do sistema.

De acordo com o detalhado na Seção 3.2.1, o `Kerberos` oferece a possibilidade de criação de chave secreta compartilhada com diversos tipos de configurações, dentre os vários algoritmos possíveis, como é possível verificar na Tabela 4.1:

Tabela 4.1: Tipos de Criptografia do Kerberos.

Família	**Cifras**	**Força**
DES	des-cbc-crc, des-cbc-md5, des-cbc-md4	Fraca
3DES	ddes3-cbc-sha1	Forte
AES	aes256-cts-hmac-sha1-96, aes128-cts-hmac-sha1-96	Forte

[2]http://docs.openafs.org/AdminGuide.pdf
[3]http://docs.openafs.org/QuickStartUnix.pdf
[4]http://web.mit.edu/kerberos/krb5-current/doc/admin/install.html
[5]Em computação, é comum usar o termo "VM"(*Virtual Machine*) para máquina virtual.

Ocorre que a versão padrão 1.6.20 do `OpenAFS`, utilizada neste trabalho e atualmente disponibilizada no repositório *stable*[6] do Debian, ainda não está preparada para a utilização de cifras AES, manipuladas pelo `Kerberos` 5. Em função desta limitação, há a necessidade de alterar as configurações padrão do `Kerberos`, de modo que este aceite pedidos de autenticação com chaves DES, relativamente mais fracas se comparadas com chaves AES.

Nenhum usuário se autentica com esta chave. Entretanto, ela é usada para criptografar os *tickets* que o KDC concede aos clientes do `OpenAFS` para apresentação aos processos do servidor de arquivos durante a autenticação mútua, como foi detalhado na Subseção 3.2.1.

A Figura 4.2 ilustra o processo de criação da chave secreta DES a ser utilizada para a célula do ambiente proposto:

```
root@afs1:/tmp# kadmin -p root/admin
Authenticating as principal root/admin with password.
Password for root/admin@UNB.BR:
kadmin:  ktadd -k /tmp/afs.keytab -e des-cbc-crc:normal afs
Entry for principal afs with kvno 2, encryption type des-cbc-crc added to keytab WRFILE:/tmp/afs.keytab.
kadmin:
```

Figura 4.2: Criação da chave simétrica no Kerberos para uso na célula do OpenAFS.

Desmembrando as partes da chave simétrica DES no modo CBC com *checksum* CRC-32 gerada pelo `Kerberos`, é possível observar que a mesma possui as seguintes características:

- DES: O *Data Encryption Standard* é um algoritmo de chave simétrica que utiliza chaves de tamanho 64 bits, dos quais 8 são usados para garantia de integridade. O tamanho efetivo das chaves é, portanto, de 56 bits;
- CBC: O módulo *Cipher Block Chaining* é utilizado para alterar o algoritmo DES, fazendo com que qual cada bloco de texto simples sofre uma operação de XOR-ed[7] com o bloco de texto cifrado anterior antes da criptografia. Desta forma, dois blocos de texto simples idênticos criptografam dados de maneira diferente; e
- CRC: Acrônimo de *Cyclic Redundancy Check*, é utilizado como método de detecção de erros para fornecer garantia de integridade dos dados.

[6] Conjuntos de *softwares* armazenados em uma árvore de diretório especial que pode ser acessada por clientes que baixam e instalam os pacotes disponibilizados. O repositório *stable* contém apenas as versões de *softwares* testadas e estáveis, sendo por este motivo o repositório mais indicado para instalação de servidores.

[7] XOR-ed

4.2 Execução de testes

Esta Seção apresenta a execução dos testes de segurança e os resultados obtidos em cada uma das características observadas, conforme planejado na Subseção 1.4.3.

4.2.1 Quanto à disponibilidade dos dados

A Subseção 3.2.3 apresentou de modo sucinto o recurso de criação de cópias de segurança operacionalizado nativamente pelo `OpenAFS` . Com a finalidade de criar cópias de *backup* de volumes do `OpenAFS` para possibilitar a restauração de dados eventualmente perdidos ou corrompidos, o sistema disponibiliza o *AFS Backup System*, recurso que passa a ser analisado em detalhes a partir das observações colhidas no ambiente virtualizado.

A execução do processo de *backup* e de restauração dos dados demanda um planejamento minucioso que envolve determinar as principais características dos dispositivos envolvidos, tais como:

- propriedades das fitas utilizadas, como capacidade, fabricante, identificação, dentre outros;
- quais são os privilégios administrativos a serem concedidos aos operadores de *backup*;
- quais volumes ou coleções de volumes deverão ser alvo do *backup*; e
- estrutura lógica de definição da hierarquia entre os diferentes tipos de *backup* (completo, diferencial).

A criação de cópias de segurança no `OpenAFS` depende, inicialmente, da especificação do grupos de volumes chamados de *volume sets*, definidos com o programa `backup addvolset`. Com os grupos definidos, o servidor, partição e nomes dos volumes devem ser configurados com a execução do programa `backup addvolentry`. Naturalmente, é possível remover volumes ou grupos inteiros de volumes, usando os programas `listvolsets` para exibir o índice de entradas de volumes e executando o programa `delvolentry` para remover a entrada desejada.

De acordo com IBM [5], é mais eficiente incluir um número limitado de volumes em uma dada entrada, uma vez que a execução de *dumps* com um grande número de volumes pode levar muito tempo para sua conclusão, acarretando aumento da probabilidade de ocorrência de falhas.

Um recurso importante identificado nos testes com o *Backup System* do `OpenAFS` é o agendamento de reciclagem de mídias, método que permite definir um tempo de expiração dos *dumps* e possibilita estabelecer o limite de tempo passado que o *backup* deve alcançar, a partir do qual os dados poderão ser sobrescritos com novas versões. A

manipulação destes parâmetros é realizada pela execução do programa `backup setexp` com o argumento `-expires`, que define a data de expiração. Como citado no início desta Subseção, este recurso necessita de definições que são estipuladas na fase de planejamento do processo de *backup*.

A Tabela 4.2 resume os principais comandos dos programas usados na configuração e manipulação do processo de *backup* no `OpenAFS`:

Tabela 4.2: Comandos dos programas de *backup*.

Comando	**Função**
`backup status`	Exibe o estado do dispositivo de *backup*
`backup dump`	Realiza o *backup* dos dados
`backup dumpinfo`	Exibe os *dumps* gravados
`backup volinfo`	Exibe o histórico dos *dumps* de volumes
`backup volrestore`	Executa a recuperação de um volume
`backup volsetrestore`	Executa a recuperação de um grupo de volumes
`backup dbverify`	Verifica a integridade da base de dados de *backup*
`backup savedb e restoredb`	Repara dados corrompidos na base de dados de *backup*
`backup deletedump`	Remove *dumps* da base de dados de *backup*

Para além da possibilidade de execução de cópias de segurança, o `OpenAFS` oferece o recurso de replicação de volumes, como foi discutido na Subseção 3.1.3. A utilização deste processo aumenta a disponibilidade dos dados, já que se um dado servidor falhar, o usuário poderá acessar o volume replicado em outra máquina.

Como apresentado na Seção 3.2.3, o `OpenAFS` apresenta-se como um sistema de arquivos que combina o conceito de Sistema de Arquivos Distribuído com recursos de *backup* e suporte a *failover*. Utilizando-se do conceito de células, conforme descrito anteriormente, o `OpenAFS` possibilita a transferência automática do acesso a seus de sistemas de arquivos inteiros de um servidor para outro em caso de erros ou falhas, sem afetar sua disponibilidade para os clientes, de modo transparente.

Diferente do recurso de cópias de segurança, que oferece garantias de recuperação de volumes de qualquer natureza, a replicação aplica-se, preferencialmente, a volumes que não mudam com frequência, como por exemplo executáveis ou volumes do tipo somente-leitura específicos do `OpenAFS`. De acordo com IBM [5], os volumes `root.afs` e `root.cell` são fortes candidatos ao processo de replicação.

Essa indicação justifica-se, uma vez que o Gerenciador de Cache do `OpenAFS` utiliza os diretórios correspondentes a estes volumes para a correta interpretação dos caminhos de diretórios do sistema. Além disso, a indisponibilidade destes volumes (`root.afs` e `root.cell`) pode causar a queda na disponibilidade de todos os outros volumes da célula, mesmo que o servidor que os armazena esteja em funcionamento [5].

A célula, base do ambiente, comporta-se como uma árvore de diretórios, com hierarquia de arquivos e pastas. Esta arquitetura torna possível o crescimento da quantidade de dados de uma célula, e a adição de novos servidores replica seus volumes, mantendo constante a visão do usuário sobre a árvore de diretórios do `OpenAFS`, provendo alta disponibilidade.

Para identificar o comportamento da replicação de volumes no ambiente virtualizado alvo de análise desta dissertação, foi necessário utilizar a ferramenta `vos addsite` para adicionar uma entrada na Base de Dados de Localização de Volumes - Volume Location Database (VLDB) referente a uma partição específica em um segundo servidor. O comando `vos release` foi aplicado para replicar/clonar o volume de origem, o que distribui a réplica para o servidor anteriormente definido. A Figura 4.3 mostra os passos para a configuração do recurso:

```
root@afs1:~# vos addsite -server afs.unb.br -partition /vicepa -id admin.afs
root@afs1:~#
root@afs1:~# vos release teste
root@afs1:~#
root@afs1:~# vos examine teste
   teste                             536870981 RW   50000 K On-line
        afs1.unb.br /vicepa
        RWrite 5360870981   ROnly 536870982   Backup 536870983
        MaxQuota      40000 K
        Creation    Sat Feb 16 07:02:18 2019
        Last Update Sun Feb 17 18:49:50 2019
        0005 accesses in the past day (i.e., vnode references)
        RWrite: 5258540981   ROnly: 593687982   Backup: 753980983
        number of sites -> 1
           server afs.unb.br partition /vicepa RO Site
        Volume is currently LOCKED
root@afs1:~#
```

Figura 4.3: Configuração de replicação de volumes entre dois servidores.

Nesta mesma imagem é possível verificar a saída do comando `vos examine`. Ele foi utilizado para exibir informações da Volume Location Database (VLDB) e do *header* do volume especificado, as quais revelam que o volume `teste`, presente no servidor `afs1.unb.br`, é replicado na partição `/vicepa` do servidor `afs.unb.br`, que é o segundo servidor `OpenAFS` da célula configurada no ambiente simulado nesta análise, conforme é possível verificar na Figura 4.1.

É relevante observar que o comando `vos release` replica o conteúdo do volume de origem, que normalmente tem atributos de leitura e escrita, para o segundo servidor indicado, mas com atributo de somente leitura, conforme é possível verificar na Figura 4.3.

4.2.2 Quanto à integridade dos dados

A análise da garantia de integridade dos dados, provida pelo `OpenAFS`, foi realizada por meio da execução do programa `backup dbverify`. Este utilitário verifica as bases de

dados do sistema em busca de dados danificados, inválidos ou corrompidos. Nos casos em que tais problemas são identificados, existe outra ferramenta do próprio sistema que executa a reparação dos danos automaticamente, incluindo, ainda, a possibilidade de remoção de blocos órfãos na memória do servidor de *backup*, liberando espaço [5].

A Figura 4.4 ilustra o comando utilizado no ambiente de testes para executar a verificação da integridade do Banco de Dados de *Backup* do servidor `afs1.unb.br`. A saída padrão do comando `backup dbverify` indica se este Banco está danificado, ou seja, se há dados corrompidos, ou se a base de dados não apresenta problemas de integridade:

```
root@afs1:~# backup dbverify -detail -localauth -cell unb.br
Database OK
   Orphan blocks 0
   Database checker was afs1.unb.br
root@afs1:~#
```

Figura 4.4: Verificação de integridade da base de dados de *Backup* do OpenAFS.

Como é possível observar, o diagnóstico do Banco de Dados de *Backup* do servidor `afs1.unb.br` foi positivo, não apresentando problemas de integridade de seus dados. Fosse o resultado diferente, ou seja, caso apresentasse indicações de dados corrompidos, seria necessário executar a ação de recuperação de dados com a ferramenta `backup savedb`, de modo a reparar o Banco de Dados.

4.2.3 Quanto à confidencialidade e ao processo de autenticação

De acordo com o exposto na Subseção 1.2, a confidencialidade garante que a informação não deve ser disponibilizada ou divulgada a indivíduos, entidades ou processos não-autorizados.

No âmbito deste trabalho, a confidencialidade naturalmente está ligada ao processo de autenticação, devendo fornecer "a segregação necessária para a informação e garante que apenas os usuários autorizados tenham acesso aos dados", conforme citado na Subseção 2.5.1.

O processo de autenticação entre cliente e servidor `OpenAFS`, abordado na Subseção 3.2.1, envolve o `Kerberos`, um serviço de autenticação de entidades (pessoas, *hosts*) que utiliza criptografia simétrica - de chave privada compartilhada - em uma rede na qual, supostamente, os dados em tráfego podem ser lidos ou modificados [53].

Realizado por meio do fornecimento de *tickeets* e *tokens*, o processo somente autoriza o acesso aos arquivos do `OpenAFS` a usuários que possuam duas autenticações válidas:

- no sistema operacional que dá acesso ao sistema de arquivos local, pois é através deste SO que o acesso ao *filespace*[8] do `OpenAFS`, por meio do Gerenciador de Cache, conforme citado na Subseção 3.1.2 ; e
- no KDC do `Kerberos`, que fornece as credenciais para o acesso ao Gerenciador de Cache.

Para analisar o processo de autenticação do `OpenAFS` no `Kerberos`, foi necessário visualizar os detalhes dos pacotes de dados trafegados entre as requisições do cliente e as respostas tanto `Kerberos` quanto do servidor `OpenAFS` durante as diversas fases do processo de autenticação.

Este processo, já definido na Seção 2.5.2, é chamado de *eavesdropping*. Na prática, a ação realizada foi a execução de intrusão passiva com a utilização de ferramenta específica para captura de tráfego de redes, de modo a permitir a coleta dos dados em tráfego e sua posterior visualização e análise gráfica.

O programa `tcpdump`[9] foi a ferramenta adotada para capturar o tráfego de dados. De acordo com Mota Filho [66], o `tcpdump` é um poderoso analisador de tráfego em modo texto, baseado na API `libcap`[10], que mostra as conexões estabelecidas pelo *host* e o tráfego correspondente.

Para executar a visualização gráfica das informações coletadas, utilizou-se o programa `Wireshark`[11]. Também baseado na API `libcap`, a aplicação possibilita, ainda acordo com Mota Filho [66], a análise de tráfegos de rede em ambiente gráfico, sendo capaz de ler arquivos gerados pelo `tcpdump`.

Destarte, a utilização conjunta destas duas ferramentas possibilitou a observação das características técnicas do sistema de autenticação usado pelo `OpenAFS`, provido pelo `Kerberos`.

A Figura 4.5 ilustra o ambiente em que foi reproduzido o ataque, cujos endereços IP serão objetos de análise no decorrer desta Subseção:

[8] Denominação do espaço de arquivos (ou árvore de diretórios) do `OpenAFS`.

[9] http://www.tcpdump.org/

[10] Esta API padroniza o formato básico dos dados capturados, permitindo que diversos aplicativos consigam identificar e manipular as informações registradas.

[11] https://www.wireshark.org/

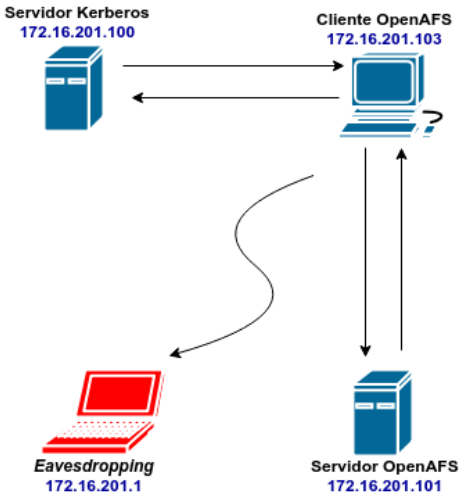

Figura 4.5: Monitoração do tráfego por *eavesdropping*.

Como é possível observar, o *host* hachurado em vermelho, conectado ao mesmo segmento da rede *ethernet* que interliga cliente e servidores `OpenAFS` e `Kerberos`, faz o papel de usuário malicioso e passa a utilizar sua interface de rede em modo promíscuo, que o possibilita utilizar recursos para registrar todo o tráfego passante.

Esta ação, caracterizada como ameaça à segurança de redes em ambientes corporativos, configura a intrusão passiva chamada de *eavesdropping*, comentada na Seção 2.5.2 e que, como salientado, possibilita o registro dos dados em trânsito na rede.

Cabe detalhar, por oportuno, que a configuração da interface de rede do *host* malicioso no modo promíscuo é condição necessária para permitir a recepção de todos os dados em trânsito na rede, já que, em modo normal de operação, uma interface somente recebe dados a ela endereçados ou, naturalmente, por ela transmitidos.

No modo promíscuo, todos os *frames*[12], independente de sua origem ou destino, são visualizados e podem ser registrados em arquivos para posterior exame, em um processo denominado como "análise de tráfego" [66].

Antes da apresentação e detalhamento das capturas do tráfego do ambiente propriamente ditas, vale examinar a Figura 4.6, que ilustra as fases da autenticação entre um dado cliente e um servidor `OpenAFS` que, como já citado, emprega o `Kerberos` como terceiro ente confiável para validação das credenciais de acesso:

[12]Unidade básica de dados que encapsula as informações que devem ser transmitidas através de dispositivos conectados por uma rede *ehernet*.

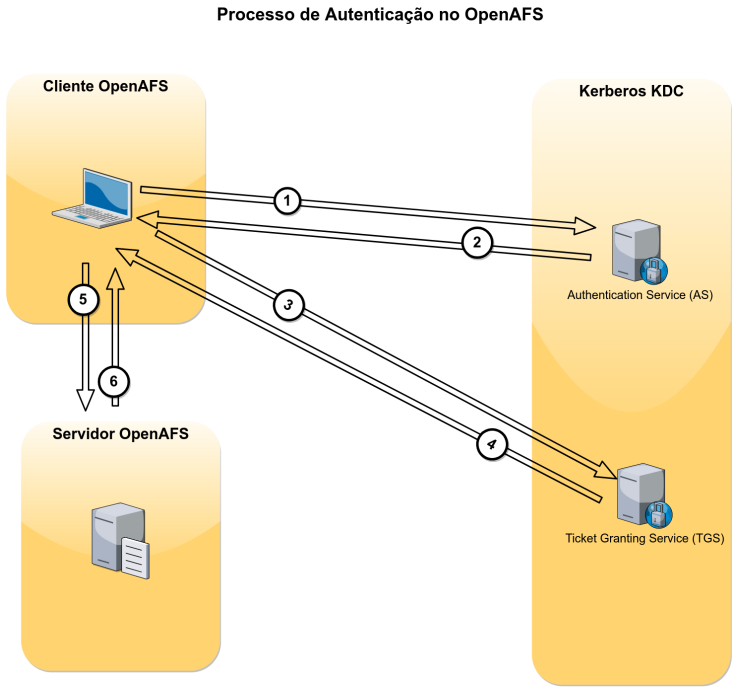

Figura 4.6: Sequência de passos da autenticação [5].

A seguir são detalhados os passos constantes da Figura 4.6:

1. o cliente `OpenAFS` envia seu ID de usuário em uma mensagem em claro para o AS. Esta mensagem não inclui a senha do cliente nem sua chave secreta (detalhado no ambiente virtualizado na Figura 4.10 adiante);
2. o AS verifica se o cliente está no banco de dados de usuários. Se sim, gera uma chave secreta para o cliente via *hash*[13]. Em seguida, o AS envia uma chave de sessão para o cliente, criptografada com a chave secreta criada anteriormente;
3. o cliente decifra a chave de sessão e envia para o TGS uma mensagem de solicitação (contendo o TGT e o ID do serviço a ser acessado) e uma mensagem do autenticador (contendo o ID do cliente e o carimbo de hora, criptografados com a chave do cliente) para o TGS;
4. o TGS decifra o TGT na mensagem de solicitação para recuperar a chave de sessão cliente / TGS e decifra a mensagem do autenticador. O TGS verifica se o cliente está autorizado a acessar o `OpenAFS` e envia um *ticket* de serviço e uma chave de sessão cliente / servidor criptografada com a chave de sessão para o cliente;

[13]Também chamada de função resumo, é uma função criptográfica bijetora que tem como domínio um conjunto de elementos de tamanho variável e como imagem um conjunto de elementos de tamanho fixo.

5. o cliente `OpenAFS` envia o *ticket* de serviço e uma nova mensagem do autenticador criptografada com a chave de sessão cliente / servidor para o servidor `OpenAFS` a ser acessado; e

6. o servidor `OpenAFS` decifra o *ticket* de serviço para recuperar a chave de sessão cliente / servidor e, em seguida, decifra a mensagem do autenticador para recuperar o registro de data e hora do cliente. O servidor `OpenAFS` envia uma mensagem de confirmação de serviço, incluindo o registro de data e hora, criptografada com a chave de sessão cliente / servidor de volta ao cliente `OpenAFS`. A partir deste ponto, cliente e servidor `OpenAFS` têm a comunicação estabelecida.

Uma vez detalhados os passos teóricos do processo de autenticação entre cliente e servidores, passa-se à descrição da fase de coleta dos dados no ambiente de testes. Como já citado, a captura dos dados foi realizada com a utilização do `tcpdump`, usando os endereços IP dos *hosts* envolvidos na simulação, conforme ilustrado na Figura 4.5.

Esta captura visa obter dados ligados ao processo de autenticação do cliente, conforme detalhado na Subseção 3.2.1 e ilustrado na Figura 3.5. A Figura 4.7 apresenta os comandos executados no *host* `172.16.201.1` para uso do `tcpdump`, de modo a possibilitar o registro dos dados passantes por sua interface de rede:

```
root@ubu:/home/borges# tcpdump -i vmnet8 -netA -vvv -w /home/borges/cliente.pcap host 172.16.201.103 and ! arp and ! port 53
tcpdump: listening on vmnet8, link-type EN10MB (Ethernet), capture size 262144 bytes
Got 19
```

Figura 4.7: Captura de tráfego no *host* 172.16.201.1.

O primeiro comando ilustrado na Figura 4.7 coloca a interface de rede “vmnet8”, que está conectada ao segmento analisado, em modo promíscuo. Examinando a sintaxe utilizada no segundo comando, é possível identificar que o `tcpdump` foi configurado para: monitorar a mesma interface; não realizar a resolução de nomes dos *hosts* e de portas (-n); mostrar dados da camada de enlace (-e); omitir dados de data e hora (-t); registrar o conteúdo dos *payloads* (-A); aumentar a quantidade de informações extraídas dos cabeçalhos dos pacotes (-vvv); gravar a captura no arquivo “/home/borges/cliente.pcap” (-w); registrar somente dados que contenham o endereço `172.16.201.103` (`OpenAFS` cliente); e excluir tanto pacotes que contenham tanto o protocolo ARP quanto tráfego de DNS, de modo a sanitizar o conteúdo capturado e facilitar sua visualização.

A captura foi executada precisamente no decorrer do processo de autenticação entre o cliente e o servidor `OpenAFS` que, sabe-se, usa o `Kerberos` como terceiro ente confiável.

A utilização dos parâmetros citados resultou no arquivo "cliente.pcap"[14] que, a partir deste ponto, será examinado com o auxílio do `Wireshark`, empregado para ler o arquivo "cliente.pcap".

A Figura 4.8 ilustra o processo de autenticação no *host* "`client`" `(172.16.201.103)`:

```
root@client:~# kinit -p useralfa@UNB.BR
Password for useralfa@UNB.BR:
root@client:~#
root@client:~# klog.krb5 -principal useralfa -cell unb.br -k UNB.BR
Password for useralfa@UNB.BR:
root@client:~# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: useralfa@UNB.BR

Valid starting       Expires              Service principal
17-02-2019 19:17:10  18-02-2019 05:17:10  krbtgt/UNB.BR@UNB.BR
        renew until 18-02-2019 19:17:05
root@client:~#
root@client:~# tokens

Tokens held by the Cache Manager:

User's (AFS ID 20001) tokens for afs@unb.br [Expires Feb 18 05:17]
   --End of list--
root@client:~#
```

Figura 4.8: Processo de autenticação e visualização de *tokens* no cliente.

Inicialmente, o comando `kinit -p useralfa@UNB.BR`[15] conecta-se ao servidor de autenticação, apresentando o usuário `useralfa` ao Realm `UNB.BR`.

A segunda parte da autenticação utiliza o comando `klog.krb5 -principal useralfa -cell unb.br -k UNB.BR`[16], que conecta o usuário `useralfa` ao servidor de autenticação, informando a célula do `OpenAFS` à qual necessita de acesso `(unb.br)`, que por sua vez está cadastrada no *Realm* `UNB.BR`.

Como é possível observar na Figura 4.8, a listagem dos *tickets* efetuada pelo comando `klist`[17] mostra que o TGT do `Kerberos` concedeu um *ticket* ao usuário, e o comando `tokens`[18] retorna a listagem dos *tokens* do usuário de `ID 20001 (useralfa)`. Como já comentado, os processos do `OpenAFS` exigem que seus clientes apresentem um *token* como evidência de que eles foram autenticados na célula local do servidor.

[14] A extensão ".pcap" é um acrônimo para a expressão *packet capture* e é comumente usada para arquivos que contêm dados de tráfegos de rede.

[15] `kinit` é um programa cliente do `Kerberos` que obtém e armazena um *ticket* inicial de um Principal armazenado em um servidor.

[16] Interface cliente do `OpenAFS` que solicita autenticação no `Kerberos` e obtém *tokens*.

[17] Lista o conteúdo do *cache* de *tickets* no cliente.

[18] Retorna os tokens recebidos do `Kerberos`

Observando-se os logs do Servidor `Kerberos` ilustrados na Figura 4.9, é possível perceber as fases da concessão do *ticket* e do *token*:

```
Feb 17 19:17:05 kerb krb5kdc[854]: AS_REQ (8 etypes {18 17 20 19 16 23 25 26}) 172.16.201.103: NEEDED_PREAUTH: useralfa@UNB.BR for krbtgt/UNB.BR@UNB.BR, Additional pre-authe
ntication required
Feb 17 19:17:10 kerb krb5kdc[854]: AS_REQ (8 etypes {18 17 20 19 16 23 25 26}) 172.16.201.103: ISSUE: authtime 1550441830, etypes {rep=18 tkt=18 ses=18}, useralfa@UNB.BR for
 krbtgt/UNB.BR@UNB.BR
Feb 17 19:17:17 kerb krb5kdc[854]: AS_REQ (11 etypes {18 17 20 19 16 23 25 26 1 3 2}) 172.16.201.103: NEEDED_PREAUTH: useralfa@UNB.BR for krbtgt/UNB.BR@UNB.BR, Additional pr
e-authentication required
Feb 17 19:17:22 kerb krb5kdc[854]: AS_REQ (11 etypes {18 17 20 19 16 23 25 26 1 3 2}) 172.16.201.103: ISSUE: authtime 1550441842, etypes {rep=18 tkt=18 ses=18}, useralfa@UNB
.BR for krbtgt/UNB.BR@UNB.BR
Feb 17 19:17:22 kerb krb5kdc[854]: TGS_REQ (11 etypes {18 17 20 19 16 23 25 26 1 3 2}) 172.16.201.103: LOOKING_UP_SERVER: authtime 0,  useralfa@UNB.BR for afs/unb.br@UNB.BR,
 Server not found in Kerberos database
Feb 17 19:17:22 kerb krb5kdc[854]: TGS_REQ (11 etypes {18 17 20 19 16 23 25 26 1 3 2}) 172.16.201.103: LOOKING_UP_SERVER: authtime 0,  useralfa@UNB.BR for afs/unb.br@UNB.BR,
 Server not found in Kerberos database
Feb 17 19:17:22 kerb krb5kdc[854]: TGS_REQ (11 etypes {18 17 20 19 16 23 25 26 1 3 2}) 172.16.201.103: ISSUE: authtime 1550441842, etypes {rep=18 tkt=1 ses=1}, useralfa@UNB.
BR for afs@UNB.BR
```

Figura 4.9: Log de concessão de *ticket* e *token* pelo `Kerberos`.

Cabe observar a exatidão nos horários tanto de cliente quanto de servidor. Como foi comentado na Subseção 3.2.3, a necessidade de sincronia dos relógios das máquinas é condição necessária para o correto funcionamento do `Kerberos`.

Esta necessidade é ratificada por Stallings [45], que destaca que quando um usuário deseja obter um *ticket*, é necessário enviar ao AS um bloco de pré-autenticação contendo um fator de confusão aleatório, um número de versão e uma estampa de tempo *(timestamp)*, que são cifrados com a chave baseada em sua senha.

O AS, então, decifra o bloco mas não envia um *ticket* de concessão de *tickets* a menos que a estampa de tempo no bloco de pré-autenticação esteja dentro de um intervalo de tempo que considere a variação de relógios e atrasos da rede.

Como será observado na apresentação das capturas de tráfego a seguir, o *timestamp* será um dado constantemente trocado entre cliente e servidor, apresentando-se como mais um recurso para integrar o processo de garantia da confidencialidade dentro do processo de autenticação.

Deste ponto em diante serão apresentadas telas do programa `Wireshark`, que contêm os dados dos pacotes trafegados na rede - entre cliente e servidor - durante o processo de autenticação, acompanhadas de comentários a respeito das informações nelas exibidas.

Inicialmente, é importante sublinhar que o Servidor `OpenAFS` não se comunica diretamente com o `KDC`. Os *tickets* de serviço, mesmo empacotados pelo *TGS*, chegam ao servidor somente através do cliente que deseja acessá-lo.

A Figura 4.10 ilustra o início de um processo de autenticação, no qual o *host* `172.16.201.103 (client)` se conecta ao *host* `(172.16.201.100) (Kerberos)`. O tipo de mensagem deste pacote, `krb-as-req`, é um acrônimo para *Kerberos Authentication Service Request*:

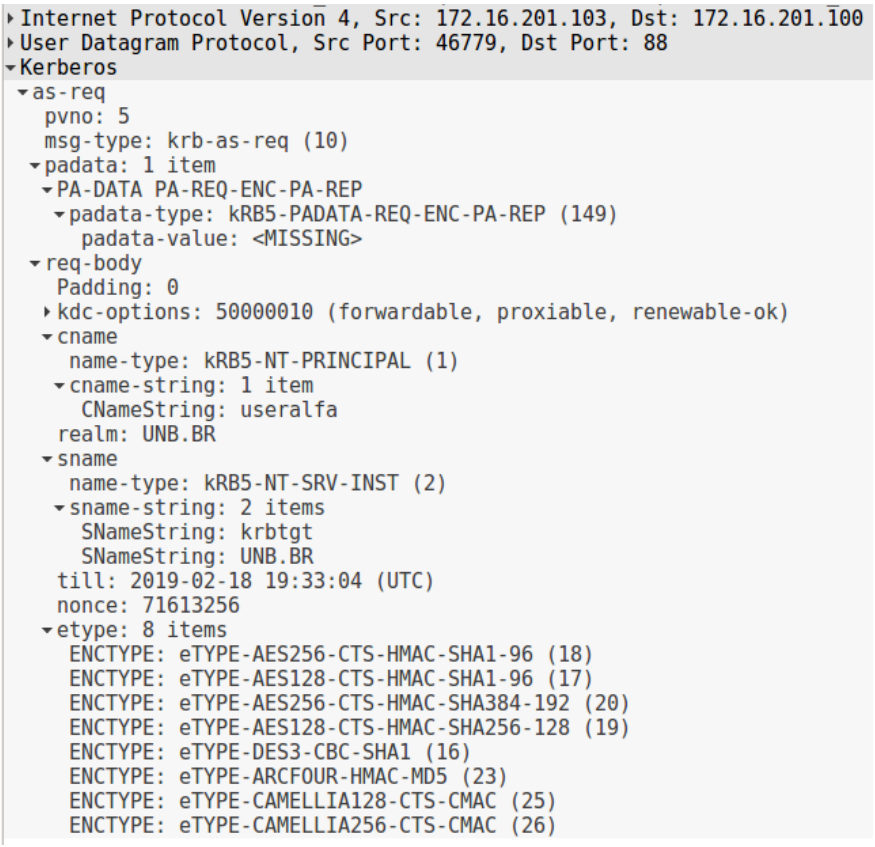

Figura 4.10: Pacote 1 - Requisição do cliente.

Nesta fase inicial, conhecida como requisição inicial de autenticação, o cliente (programa `(kinit)` solicita ao KDC, mais especificamente ao AS, um TGT *(Ticket Granting Ticket)*. É possível verificar na Figura 4.10 que esta requisição não é criptografada. Nela, o cliente envia em texto claro sua própria identidade e a identidade do servidor para o qual está solicitando credenciais. Vale destacar os seguintes dados:

- `CNameString`: contém a parte do nome do identificador do Principal do cliente `(useralfa)`;
- `sname_string`: contém o TGT e o Realm do `Kerberos`; e
- `till`: contém a data de expiração solicitada pelo cliente.

O AS verifica a existência do usuário e cria uma resposta criptografada, com uma chave baseada na senha do usuário, de modo que apenas o usuário legítimo possa decifrá-la.

A Figura 4.11 expõe os dados do segundo pacote, com origem no *host* `172.16.201.100 (Kerberos)` com destino ao *host* `172.16.201.103 (client)`. A mensagem deste pacote é do tipo `krb-error`, que contém informações adicionais que detalham o erro identificado no servidor e informa dados que o cliente pode tomar como base para corrigi-los:

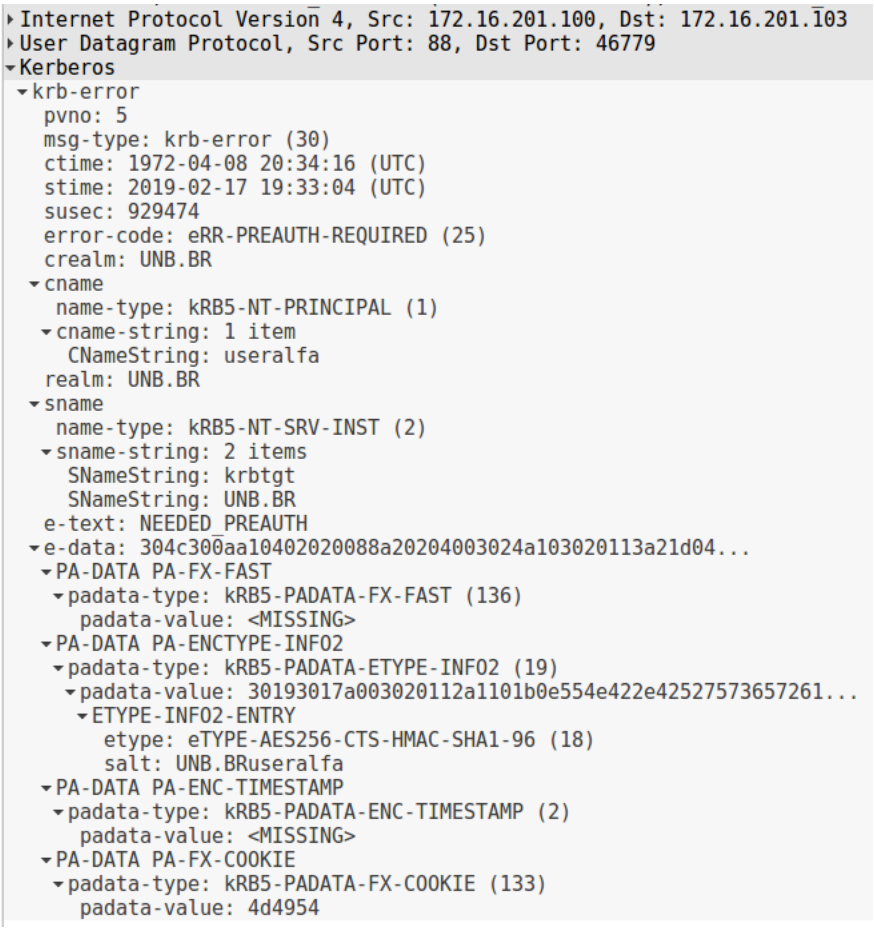

Figura 4.11: Pacote 2 - Resposta do AS.

Em resposta ao pacote 1, este pacote, apesar de transparecer um erro, na realidade é uma informação do servidor ao cliente, informando que este deve reenviar uma nova requisição, especificando um método de *timestamp* aceitável pelo servidor para a pré-autenticação. O comportamento esperado nesta situação é que o cliente envie um novo pacote com as mesmas características, porém com as informações referentes a *timestamp* configuradas.

Mais uma vez, observa-se a importância crucial da sincronia entre os relógios dos *hosts* envolvidos no processo de autenticação, conforme mencionado na Subseção 3.2.3. O cliente, enviando o *timestamp* criptografado ao servidor, possibilita ao KDC verificar se o tempo está dentro de um intervalo curto o suficiente para tornar mínimo o risco de

um ataque de repetição. Desta forma, com uma referência de tempo para expiração, o KDC pode identificar pedidos de autenticação que estejam fora deste intervalo, evitando a reutilização de *tickets* obsoletos.

A Figura 4.12 ilustra o terceiro pacote, com origem no *host* `172.16.201.103 (client)` e destino ao *host* `172.16.201.100 (Kerberos)`. O tipo de mensagem deste pacote é, também, do tipo `krb-as-req`:

Figura 4.12: Pacote 3 - Nova requisição do cliente.

Como esperado, em resposta ao pacote 2 o cliente envia o pacote 3, desta feita com um volume maior de informações. A diferença fundamental entre este pacote e o pacote 1 é a existência de dados criptografados em seu corpo, como é o caso do *timestamp*, solicitado no pacote anterior. Conforme é possível verificar no campo *cipher*, o *timestamp* é cifrado com a chave simétrica do usuário - neste caso sua própria senha - usando um algoritmo suportado pelo servidor `Kerberos` (AES256-CTS-HMAC-SHA1-96).

É importante ressalvar que, apesar da existência do recurso de controle de *timestamp*, um usuário malicioso pode tentar replicar um pedido válido, dentro da janela de tempo definida no servidor. Caso receba duas solicitações (original e replicada) com o mesmo *timestamp*, o servidor registra este evento em seus *logs*. É possível, ainda, que um atacante consiga suprimir a mensagem original, o que evitaria o registro do evento no servidor.

A Figura 4.13 traz os dados do pacote de origem no *host* `172.16.201.100 (Kerberos)` com destino ao *host* `172.16.201.103 (client)`. O tipo de mensagem deste pacote, `krb-as-rep`, é um acrônimo para *Kerberos Authentication Service Reply*. Antes de iniciar a resposta, o AS extrai a chave do cliente de sua base de dados e decifra os dados de pré-autenticação. Um destes dados é o *timestamp*:

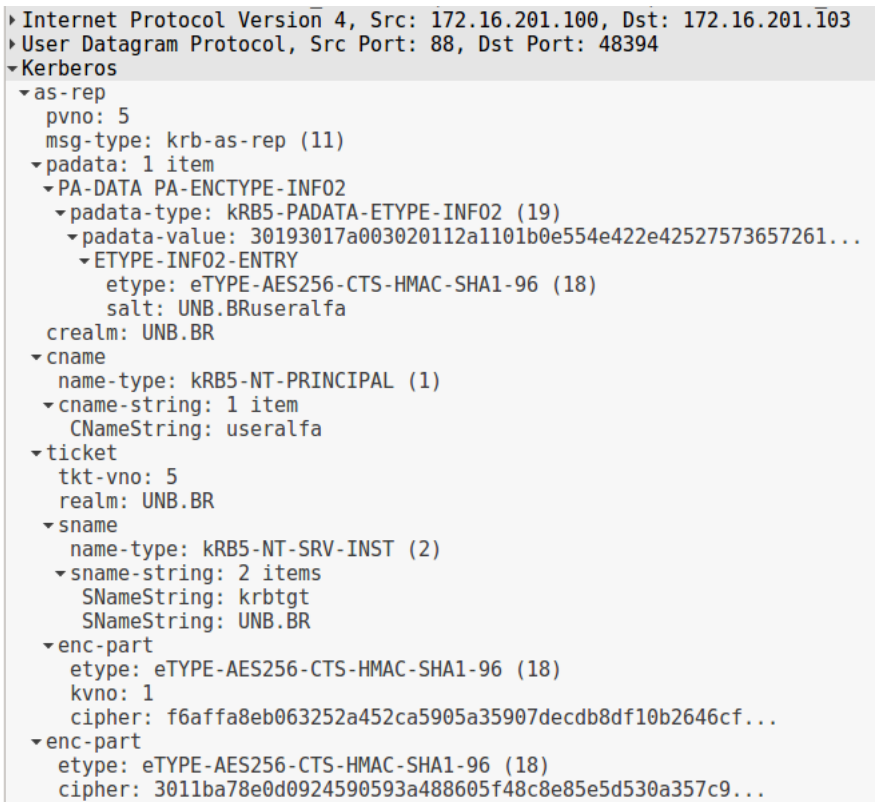

Figura 4.13: Pacote 4 - *Kerberos Authentication Service Reply*.

Estando o *timestamp* correto e dentro da janela de tempo válida, o AS assume que os dados foram cifrados com a chave autêntica do cliente e, então, gera e envia neste pacote uma chave randômica para ser usada como chave de sessão na comunicação.

Observando os campos `enc-part`, é possível verificar que eles contêm uma chave compartilhada com o `KDC`, que é a mesma que está dentro do TGT. Sendo assim, apenas o Servidor `Kerberos` pode decifrar o TGT. Ainda, usando o *ticket* e a chave de sessão, o

cliente pode, a partir de agora, solicitar um *token*, também chamado de *ticket* de serviço, para acessar o servidor `OpenAFS`.

A Figura 4.14 exibe os dados do pacote de origem no *host* `172.16.201.103 (client)` com destino ao *host* `172.16.201.100 (Kerberos)`. Esta mensagem é do tipo `tgs-req`, enviada do cliente para o *Ticket Granting Server* (TGS) solicitando um *token* (*ticket* de serviço):

```
▸ Internet Protocol Version 4, Src: 172.16.201.103, Dst: 172.16.201.100
▸ User Datagram Protocol, Src Port: 57683, Dst Port: 88
▾ Kerberos
 ▾ tgs-req
    pvno: 5
    msg-type: krb-tgs-req (12)
  ▾ padata: 2 items
   ▾ PA-DATA PA-TGS-REQ
    ▾ padata-type: kRB5-PADATA-TGS-REQ (1)
     ▾ padata-value: 6e82021c30820218a003020105a10302010ea20703050000...
      ▾ ap-req
          pvno: 5
          msg-type: krb-ap-req (14)
          Padding: 0
        ▸ ap-options: 00000000
        ▾ ticket
            tkt-vno: 5
            realm: UNB.BR
          ▾ sname
              name-type: kRB5-NT-SRV-INST (2)
            ▾ sname-string: 2 items
                SNameString: krbtgt
                SNameString: UNB.BR
          ▾ enc-part
              etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
              kvno: 1
              cipher: 6ac8b86369ea8514f26cdaa825468b11979c4f49c6b80ac4...
        ▾ authenticator
            etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
            cipher: 859b47a2ae58f90f8613f515f55c115c07577a3feab33f47...
   ▾ PA-DATA PA-FX-FAST
    ▾ padata-type: kRB5-PADATA-FX-FAST (136)
     ▾ padata-value: a081c93081c6a1173015a003020110a10e040ccac87f989c...
      ▾ armored-data
       ▾ req-checksum
          cksumtype: cKSUMTYPE-HMAC-SHA1-96-AES-256 (16)
          checksum: cac87f989c94fa648e48aaf9
        ▾ enc-fast-req
          etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
          cipher: 8de06ff9fef884e2aca1c34d6e0e4640efcb8ce5e7f5f946...
 ▾ req-body
    Padding: 0
  ▸ kdc-options: 50810000 (forwardable, proxiable, renewable, canonicalize)
    realm: UNB.BR
  ▾ sname
     name-type: kRB5-NT-PRINCIPAL (1)
   ▾ sname-string: 2 items
      SNameString: afs
      SNameString: unb.br
    till: 2019-02-18 07:03:02 (UTC)
    nonce: 1550437382
  ▸ etype: 11 items
```

Figura 4.14: Pacote 5 - Solicitação de *token* para acesso à célula “unb.br”.

Após a decifra da chave de sessão (criada no passo anterior - Figura 4.13), o cliente envia neste pacote duas informações para para o TGS: uma solicitação de *token* para acesso ao `OpenAFS`, contendo o TGT e a célula desejada `(unb.br)`; e uma mensagem de

autenticação, contendo o ID do cliente e o *timestamp*, ambos criptografados também com a chave de sessão.

A Figura 4.15 refere-se aos dados do pacote de origem no *host* `172.16.201.100 (Kerberos)` com destino ao *host* `172.16.201.103 (client)`:

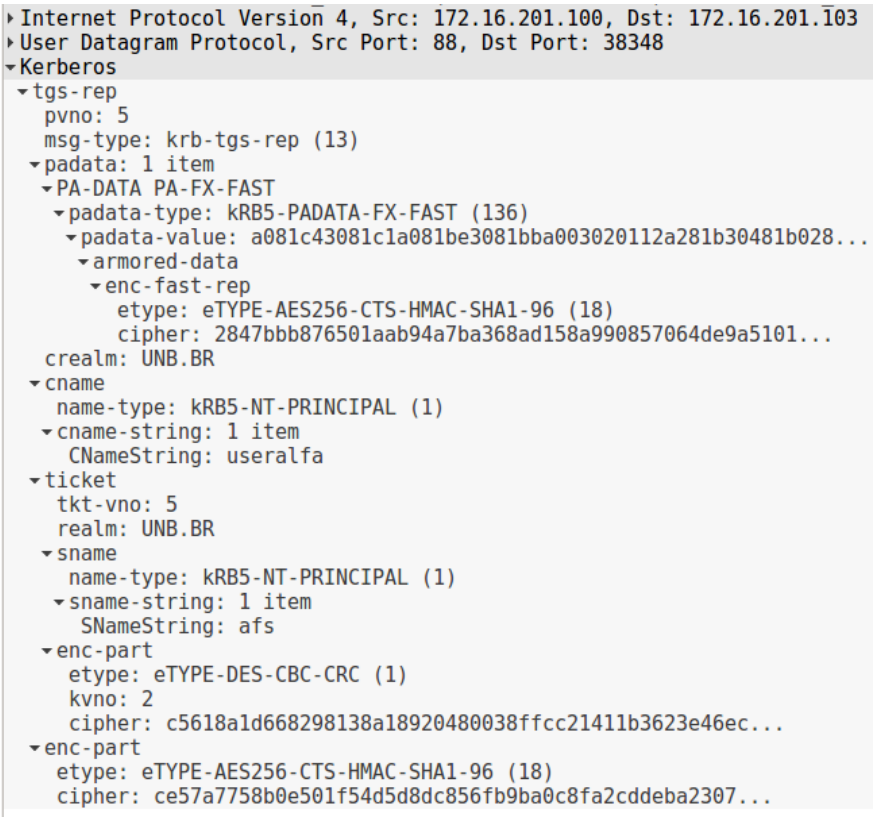

Figura 4.15: Pacote 6 - Envio do *token* para acesso à célula “unb.br”.

O TGS decifra o TGT da mensagem de solicitação (Figura 4.14) para recuperar a chave de sessão e também decifra o *timestamp*. Verifica, então, se o usuário está autorizado a acessar a célula solicitada. Finalmente, o TGS envia um *token* para acesso à célula `unb.br` do `OpenAFS`, criptografado com a chave de sessão.

Cabe ressaltar que o *ticket* de serviço também é cifrado por uma chave secreta, compartilhada entre o `KDC` e o `OpenAFS`. Por este motivo, o `OpenAFS` poderá autenticar o cliente, já que tem a certeza de que o cliente recebeu o *ticket* de serviço de um terceiro confiável (`Kerberos`).

A Figura 4.16 contém os dados do pacote de origem no *host* `172.16.201.103 (client)` com destino ao *host* `172.16.201.101 (OpenAFS)`. A partir deste ponto do processo, entra em cena o protocolo `RX`, detalhado na Subseção 3.2.1:

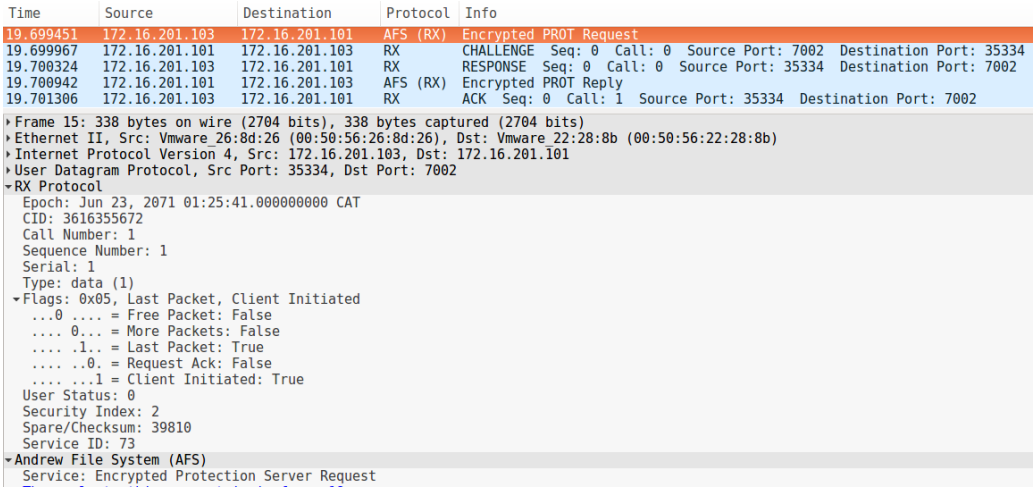

Figura 4.16: Pacote 7 - Requisição do cliente ao servidor OpenAFS.

Como é possível visualizar, o cliente envia uma requisição ao servidor, contendo a ID do serviço. Observando a porta de destino do protocolo de transporte, identifica-se a porta 7002. De acordo com [5], o serviço demandado é o `pst database`, que possui o código 73, constante do campo *Service ID*. O `pst database`, em verdade, é o `Protection Database`, serviço do `OpenAFS` que armazena informações sobre usuários, máquinas clientes e grupos em um banco de dados, conforme detalhado na Subseção 3.1.1.

Desta forma, o servidor de arquivos usa esta base para determinar se os clientes estão ou não autorizados a acessar dados das células do `OpenAFS`. Não somente esta, mas outras verificações são realizadas internamente pelo serviço, como consultas à ACL's da célula que o usuário solicita, conferência de grupos e outras [5].

É importante salientar que o cliente, quando envia esta requisição, armazena em memória um registro de *timestamp* e, juntamente com outras informações da conexão, deriva uma chave de transporte [67]. O resultado é a solicitação criptografada específica do protocolo AFS, conforme é possível verificar na Figura 4.16.

A Figura 4.17 traz os dados do pacote de origem no *host* `172.16.201.101 (OpenAFS)` com destino ao *host* `172.16.201.103 (client)`.

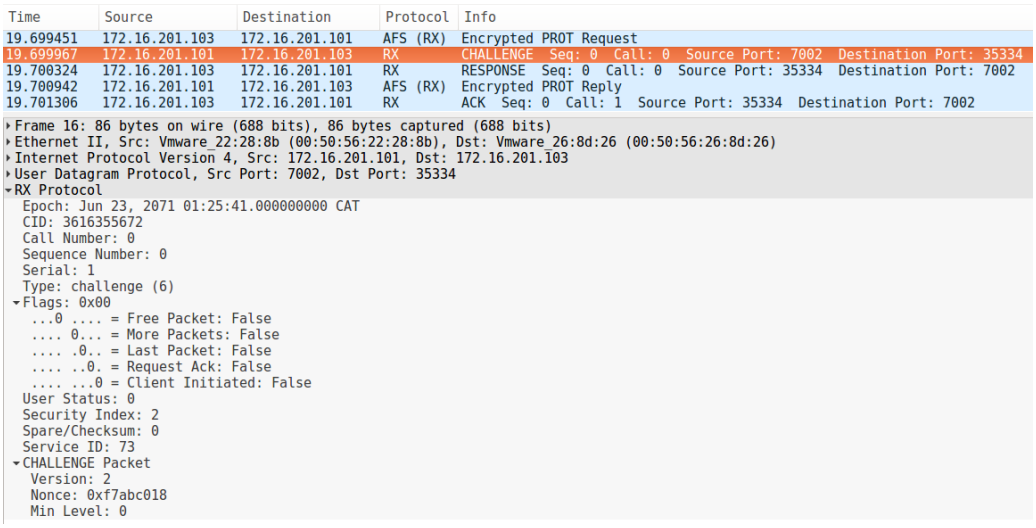

Figura 4.17: Pacote 8 - Envio do desafio do servidor OpenAFS ao cliente.

Como resposta à requisição do cliente, o servidor, seguindo o protocolo `RX`, envia um desafio ao cliente. Conforme Honeyman et al. [68], sempre que uma solicitação de acesso a um sistema de arquivos é recebida, o servidor verifica se o pedido está associado a uma conexão já autenticada. Se sim, a solicitação é atendida com as credenciais armazenadas no *cache* associado à conexão. Se não, o servidor emitirá um desafio ao gerenciador de cache no cliente, já que este está preparado para aceitar uma solicitação como esta.

O pacote de desafio consiste essencialmente em um identificador de *nonce* de 32 bits, usando as operações definidas para o objeto de segurança do protocolo `RX`, conforme estabelecido em [69]. Um *nonce* é uma sequência de caracteres aleatórios ou pseudoaleatórios, gerada por um protocolo de autenticação, que visa auxiliar no processo de garantia da não reutilização de mensagens em comunicações futuras.

Com estas características, o *nonce* auxilia na proteção aos ataques de repetição, que ocorrem quando um usuário malicioso intercepta dados e, então, pode atrasá-los ou reenviá-los de forma fraudulenta. Como a mensagem é adequadamente criptografada, o receptor pode tratar a mensagem como correta e executar ações desejadas pelo atacante.

A Figura 4.18 ilustra os dados do pacote de origem no *host* `172.16.201.103 (client)` com destino ao *host* `172.16.201.101 (OpenAFS)`:

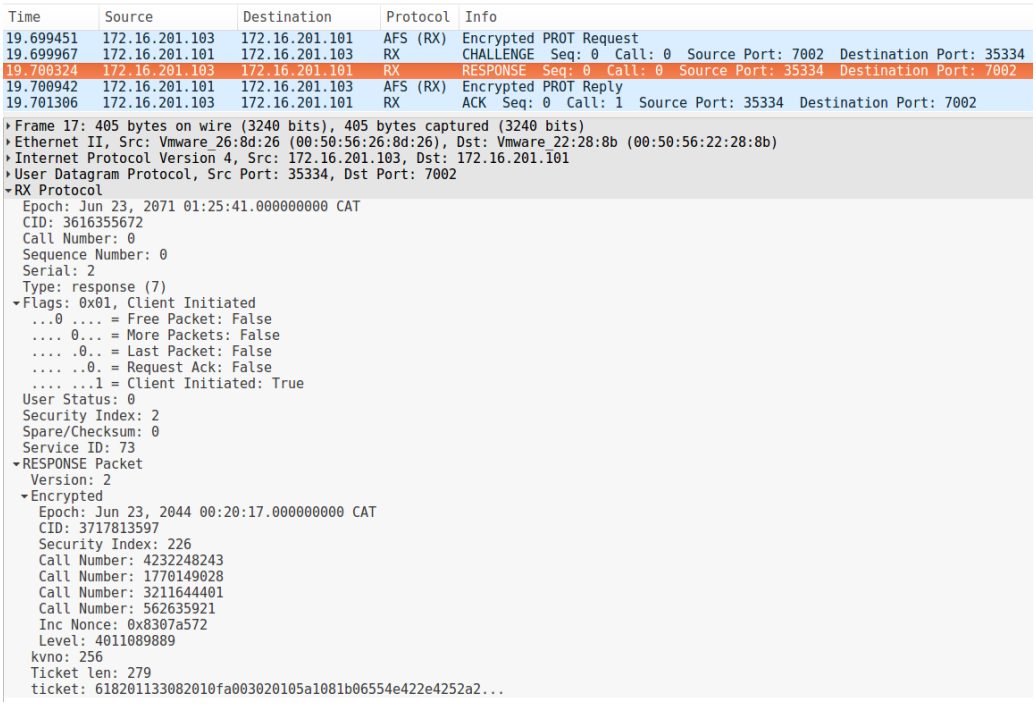

Figura 4.18: Pacote 9 - Resposta do cliente ao desafio do servidor OpenAFS.

O *payload* deste pacote é específico da camada de segurança de dados e é usado para autenticar a conexão `RX` [69]. Como resposta ao desafio, o cliente tão somente incrementa o *nonce*, cifra o *token* usando a chave de sessão e, então, envia estes dados para o servidor `OpenAFS`.

A Figura 4.20 ilustra os dados do pacote de origem no *host* `172.16.201.101 (OpenAFS)` com destino ao *host* `172.16.201.103 (client)`:

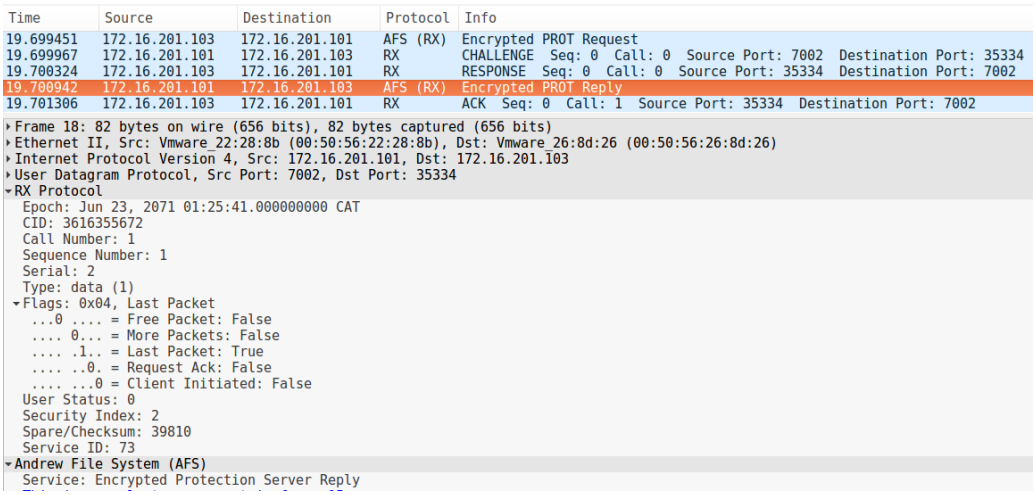

Figura 4.19: Pacote 10 - Validação da conexão pelo servidor OpenAFS.

O servidor `OpenAFS` decifra o *token* (ou *ticket* de serviço) para recuperar a chave de sessão e, em seguida, decifra a mensagem do autenticador para recuperar o *timestamp* do cliente. Então, como é possível verificar na Figura, o servidor envia a mensagem de confirmação da concessão de acesso ao serviço, incluindo mais uma vez o *timestamp* criptografado com a chave de sessão.

A Figura 4.20 ilustra os dados do pacote de origem no *host* `172.16.201.103 (client)` com destino ao *host* `172.16.201.101 (OpenAFS)`:

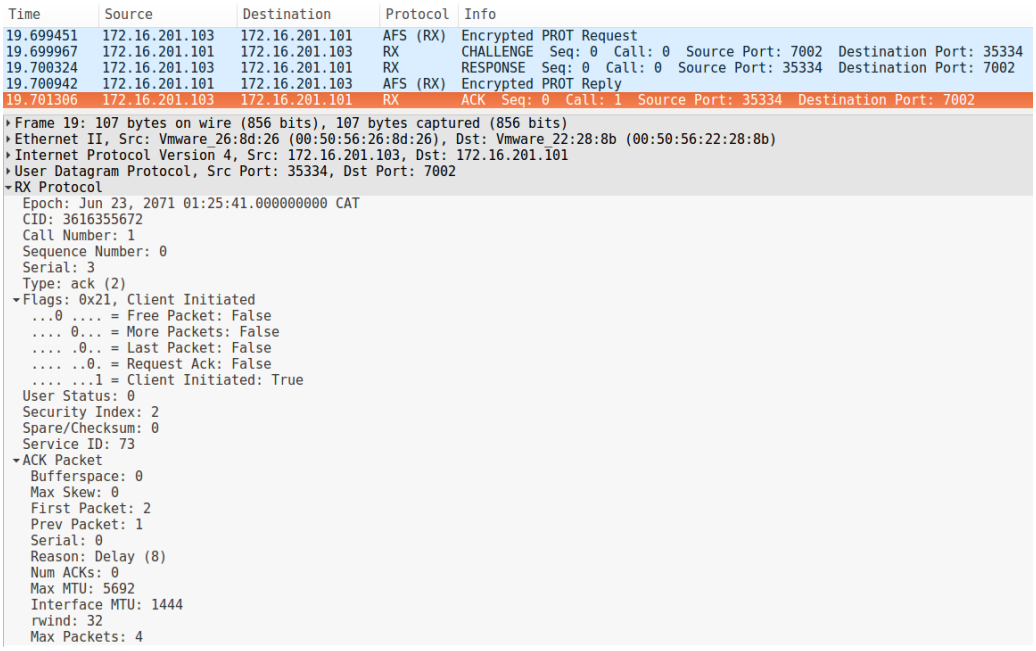

Figura 4.20: Pacote 11 - Acesso autenticado ao OpenAFS.

Finalmente, o cliente decifra a mensagem de confirmação do serviço e verifica se o *timestamp* está correto. Neste ponto a autenticação mútua está concluída, conforme comentado na Seção 3.2.1. A partir deste ponto, o cliente está autorizado a emitir solicitações de serviço normalmente ao `OpenAFS`, como é possível verificar na mensagem do tipo `ACK` acima.

4.3 Análise

Esta Seção apresenta a análise dos dados obtidos na Seção anterior.

Em complemento ao citado na Subseção 3.2.1 e analisado em detalhes na Subseção 4.2, cabe citar Bžoch and Šafařík [24], os quais reforçam que "a segurança em sistemas de arquivos distribuídos é implementada usando autenticação e controle de acesso aos arquivos".

De acordo com o estudo, "o `Kerberos` é um protocolo de autenticação de rede que permite autenticação dos usuários e impede nodos no SAD de uma comunicação de autenticação ou de autenticação repetida e fornece uma autenticação mútua: tanto o cliente quanto o servidor realizam verificação de autenticidade". Este sistema, de acordo com os autores, pode ser usado em todos os sistemas de arquivos distribuídos como protocolo de

autenticação, o que reforça a aplicabilidade do `Kerberos` como sistema de autenticação para o `OpenAFS`.

Com relação à autenticação, conforme analisado na Subseção 4.2.3, conclui-se que uma consequência natural do processo de autenticação é que o servidor conclui que o cliente está autorizado a fazer uma solicitação porque a mensagem de solicitação faz sentido quando o servidor a decifra usando a chave de sessão. Se o cliente usar uma chave de sessão diferente daquela que o servidor encontra dentro do *ticket*, a mensagem de solicitação permanecerá ininteligível mesmo após sua decifra.

Além disso, as duas cópias da chave de sessão (a de dentro do *ticket* e que o cliente usou) só podem ser as mesmas se ambas vierem do TGS. O cliente não pode fingir conhecimento da chave de sessão porque ele não pode visualizar o conteúdo do *ticket*, já que está com a chave de criptografia do servidor, conhecida apenas por ele pelo TGS. O servidor confia no TGS para oferecer *tokens* apenas a clientes com quem ele (TGS) foi autenticado, e, portanto, o servidor `OpenAFS` pode confiar na legitimidade do cliente.

Restou claro que o processo de autenticação do `Kerberos` é versátil e flexível. Todavia, ao analisar a Figura 4.15, é possível verificar que o protocolo DES-CBC-CRC é utilizado para criptografar os *tickets* que o KDC concede aos clientes do `OpenAFS` para apresentação aos processos do servidor de arquivos durante o processo de autenticação mútua, como foi detalhado na Subseção , mais especificamente visualizado na Figura 4.2.

O próprio MIT [70] ressalta que alguns tipos de criptografia, como o protocolo DES-CBC-CRC, são definidos como fracos, haja vista e, portanto, indesejáveis, sendo admitidos apenas em redes nas quais os sistemas operacionais não suportem os tipos de criptografia mais recentes.

Há várias fontes que publicaram as vulnerabilidades do DES, podendo ser citadas [71], [72] e [73], dentre inúmeras outras. A maior parte dos estudos consultados apontam os ataques de força bruta como os mais práticos e eficientes, embora haja outros ataques teóricos que supostamente têm menos complexidade do que o ataque de força bruta, como a criptoanálise diferencial, a criptoanálise linear e o *"Davies Attack"*, que envolve criptoanálise estatística.

De fato, durante os testes realizados, o `Kerberos`, por padrão, não aceitou a utilização do DES como algoritmo a ser utilizado em nenhum dos passos do processo de autenticação mútua. A maneira encontrada foi utilizar a diretiva `allow_weak_crypto = true` na seção `libdefaults` do arquivo de configuração `/etc/krb5.conf` do `Kerberos`, que controla as propriedades do sistema. Esta opção permite a operação de algoritmos considerados inseguros no mecanismo de funcionamento do `Kerberos`.

A correção para este problema de segurança abrangendo o `OpenAFS` envolve a mudança no tipo de criptografia utilizada pelo sistema, utilizando versões mais novas do OpenAFS,

que necessitam de compilação manual. Em versões como a que foi testada, é possível utilizar o estudo apresentado por Chernyakhovsky [74] para implementar a solução para o problema.

Uma forma de demonstrar o controle de acesso envolvido no processo de autenticação é tentar acessar o volume do usuário sem a correta autenticação no servidor. A Figura 4.21 ilustra duas situações distintas:

```
root@client:/# kinit -p useralfa@UNB.BR
Password for useralfa@UNB.BR:
kinit: Password incorrect while getting initial credentials
root@client:/# kinit -p useralfa@UNB.BR
Password for useralfa@UNB.BR:
root@client:/# cd /afs/unb.br/user/c/cc/useralfa/
-bash: cd: /afs/unb.br/user/c/cc/useralfa/: Permissão negada
root@client:/# klog.krb5 -principal useralfa -cell unb.br -k UNB.BR
Password for useralfa@UNB.BR:
root@client:/# klist -5
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: useralfa@UNB.BR

Valid starting       Expires              Service principal
17-02-2019 17:34:05  18-02-2019 03:34:05  krbtgt/UNB.BR@UNB.BR
        renew until 18-02-2019 17:34:01
root@client:/# cd /afs/unb.br/user/c/cc/useralfa/
root@client:/afs/unb.br/user/c/cc/useralfa# ls -lah
total 934K
drwxrwxrwx 2 root   root 2,0K fev 17 16:04 .
drwxr-xr-x 2 daemon root 2,0K fev 17 15:24 ..
-rw-r--r-- 1  20001 root 930K fev 17 16:04 openafs-modules-dkms_1.6.20-2+deb9u2_all.deb
root@client:/afs/unb.br/user/c/cc/useralfa#
```

Figura 4.21: Acesso negado e permitido ao volume do usuário.

– A primeira, em que o usuário faz uma tentativa de entrar no diretório onde está montado seu volume no `OpenAFS`, depois de entrar com credenciais inválidas junto ao `Kerberos`. É possível observar que o usuário recebe a notificação de `Permissão negada`; e

– A segunda, após realizar as duas etapas da autenticação, listando inclusive os *tokens* em seu *cache*, o usuário não encontra nenhuma restrição de acesso a seu volume, sendo-lhe permitido listar, remover ou adicionar arquivos em seu volume.

Cabe ressaltar, ainda de acordo com a Figura 4.21, que o dono do arquivo listado no diretório possui a `ID 20001`, que é a ID do usuário `useralfa`, cadastrado no `OpenAFS` no momento de sua inserção no sistema.

Com relação à disponibilidade, e a título de complemento, foi realizada uma pesquisa no CVE[19], a fim de identificar vulnerabilidades de segurança no `OpenAFS`. O último registro, datado de 11/09/2018, refere-se a uma falha ligada ao tamanho de variáveis possíveis

[19]Base de dados pública internacional voltada para troca de informações sobre falhas de segurança em *softwares*. https://cve.mitre.org/.

para entrada do RPC. No ponto em que foi analisado, o problema permitia a um usuário malicioso enviar grandes volumes de dados a um servidor, o que poderia causar um dano na disponibilidade do `OpenAFS` que se define como ataque de negação de serviço, ou *Denial of Service (DoS)*.

A falha foi reconhecida pela equipe de desenvolvimento do `OpenAFS` e corrigida no dia 28/09/2018, data em que foram lançadas as novas versões dos pacotes do OpenAFS nos repositórios do Debian Linux[20].

4.4 Considerações finais

Este capítulo apresentou a análise dos dados obtidos com a execução de observações no ambiente simulado do `OpenAFS`.

Foram abordados os aspectos ligados a disponibilidade, integridade, confidencialidade e apresentados, ainda, a arquitetura do sistema alvo dos testes, os testes realizados seguidos de seus respectivos resultados e, finalmente, a análise dos resultados obtidos.

O próximo capítulo apresentará a conclusão final da dissertação.

[20]Dados obtidos no histórico de segurança do projeto, conforme citado na Seção 3.2.

Capítulo 5

Conclusão e trabalhos futuros

Este capítulo traz a conclusão da dissertação, apresentando sua visão geral, o exame dos objetivos, a contribuição do trabalho, as limitações enfrentadas e, por fim, as sugestões para trabalhos futuros.

5.1 Visão geral do trabalho

Este trabalho levou em consideração a necessidade de implantação de um Sistema de Arquivos Distribuídos em uma organização de médio porte. Para tanto, dentre os SAD existentes, optou-se pelo `OpenAFS`, tendo em vista suas características operacionais, abordadas no Capítulo 3 deste trabalho.

Uma vez que a pesquisa bibliográfica apresentada no Capítulo 2 não revelou estudos ligados à análise de segurança específica da solução, esta dissertação propôs o exame desta característica, de modo a validar o sistema e apresentar considerações relacionadas aos conceitos formais da Segurança da Informação.

5.2 Exame dos objetivos e contribuição

Esta Seção recapitula os objetivos geral e específicos da dissertação, resumindo os resultados alcançados.

O objetivo geral, descrito na Seção 1.3.1 foi o de executar uma análise da segurança do `OpenAFS`, avaliando a disponibilidade, integridade e confidencialidade dos dados, além de avaliar a segurança de seu sistema de autenticação.

Já os objetivos específicos, apresentados na Seção 1.3.2, são aqui recordados, seguidos de seus resultados:

1. **apresentar a fundamentação teórica em forma de pesquisa bibliográfica, de modo a identificar as características dos Sistemas Distribuídos (SD) e, mais especificamente, dos Sistemas de Arquivos Distribuídos (SAD), verificando os aspectos de Segurança da Informação ligados a disponibilidade, integridade e confidencialidade dos dados manipulados pelos sistemas;**

   A pesquisa bibliográfica relacionada na Seção 2.5.2 buscou obter os mais recentes trabalhos relacionados à segurança de SAD, especialmente os ligados às características de Segurança da Informação. Com base no apresentado e de acordo com as características técnicas do `OpenAFS`, foi possível verificar que:

   (a) o `OpenAFS` utiliza um esquema de autenticação que fornece proteção contra ataques de *eavesdropping*. Os protocolos implementados protegem o tráfego de credenciais contra monitoração não-autorizada, uma vez que o fluxo de dados não trafega em claro entre cliente e servidor;

   (b) o protocolo DES-CBC-CRC é utilizado para criptografar os *tickets* que o KDC concede aos clientes do `OpenAFS` para apresentação aos processos do servidor de arquivos durante a autenticação mútua, sendo este protocolo vulnerável a ataques de força bruta; e

   (c) A tolerância a falhas discutida na Seção 4.2.1 é implementada no `OpenAFS`. De acordo com o que foi apresentado na Subseção 3.1.3, a replicação dos dados confere um mínimo de tolerância a falhas nas mídias de armazenamento do sistema.

2. **identificar diferentes tipos de SAD e comparar as características técnicas de segurança destes com o funcionamento do `OpenAFS`, a fim de particularizar a pesquisa com estudos que tratam especificamente do `OpenAFS`, com vistas a compreender os níveis de segurança que o sistema oferece aos dados nele armazenados;**

   O Capítulo 3 apresentou um estudo comparativo entre o `OpenAFS` e outros três SAD, contrastando as características de gerais, de arquitetura, segurança e cache. Foi possível concluir, da comparação, que o `OpenAFS` aplica-se a ambientes que necessitem de um sistema distribuído para compartilhamento de arquivos que ofereça escalabilidade e tolerância a falhas.

3. **executar testes com cada um dos subsistemas do `OpenAFS` que executam operações ligadas à disponibilidade, integridade, confidencialidade dos dados e de credenciais de acesso, avaliando os resultados por meio da**

sistematização dos dados coletados, de modo a permitir uma comparação com o estado da arte na área de segurança de dados em SAD.

O Capítulo 4 apresentou a execução dos testes realizados com a finalidade de aferir a capacidade dos subsistemas do `OpenAFS` relacionadas à segurança. Foram tratadas as funções do sistema relacionadas à disponibilidade, integridade, confidencialidade e sistema de autenticação do `OpenAFS`.

A contribuição deste trabalho está relacionada a descrição das características de segurança de um SAD extensivamente utilizado em grandes estruturas de compartilhamento de arquivos pelo mundo. A pesquisa bibliográfica executada não identificou, porém, estudos ou análises a respeito do sistema, relacionadas à segurança do `OpenAFS`. Desta forma, esta dissertação busca iniciar tal análise, propondo o presente estudo e vislumbrando potenciais melhorias em trabalhos futuros.

5.3 Limitações

Os óbices enfrentados no decorrer da elaboração desta dissertação foram relacionados à dificuldade de implementação da estrutura virtualizada que forma o núcleo central de operação do `OpenAFS`.

Em outras palavras, a configuração da célula `OpenAFS`, utilizada como prova de conceito para os testes, foi realizada com a utilização de virtualização baseada no *VMWare Player*, *software* de virtualização gratuito para uso pessoal, o qual permitiu a montagem do ambiente.

Em que pese o fato de que foi possível executar todas as máquinas virtuais necessárias para a realização dos testes e capturas de tráfego de rede para análise, foram observadas quedas de performance, especialmente durante a execução de operações complexas, como a execução de *backup* e a replicação de volumes.

5.4 Trabalhos futuros

Vislumba-se a possibilidade de aprofundar os testes executados na Subseção 4.2 de modo a obter mais detalhes sobre a interação em nível de rede entre o `OpenAFS` e o `Kerberos`.

Ainda, percebe-se a necessidade de complementar a análise sobre a segurança do `Kerberos`, em função da possibilidade de vulnerabilidades a diferentes métodos de ataque além do *eavesdropping*.

Um exemplo é o ataque de "*Golden Ticket*". Analisado e remediado por Plotnik et al. [75]), o *Golden Ticket* é o *token* de autenticação da conta `KRBTGT`, que é a conta

privilegiada do `Kerberos`, que tem a função de cifrar todos os *tokens* de autenticação. Um atacante que obtenha acesso a uma conta com privilégios elevados que possa acessar o controlador de domínio pode então usar técnicas para obter o *hash* de senha para a conta `KRBTGT`, ficando então livre para criar o *Golden Ticket* e utilizar os recursos da rede sem restrições.

Existem ainda outros ataques possíveis, como o "*Pass-the-ticket*", "*Encryption downgrade with Skeleton Key Malware*" e "*DCShadow attack*". Recentes estudos na área apontam possíveis fragilidades que podem ser exploradas, como por exemplo o explanado por Catota et al. [76]. Tais trabalhos podem subsidiar uma análise voltada para a manipulação de *tickets*.

Referências

[1] Axway. (2018) Axway co. [Online]. Available: https://docs.axway.com/bundle/APIGateway_762_IntegrationKerberos_allOS_en_HTML5/page/Content/KerberosIntegration/kerberos_overview.htm x, 41

[2] G. Al Sadi, "Tuning and optimizing network file system server performance," *International Journal of Computer Applications*, vol. 134, no. 10, 2016. x, 47, 48

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 29–43. [Online]. Available: http://doi.acm.org/10.1145/945445.945450 x, 48, 49

[4] A. Adamov, "Distributed file system as a basis of data-intensive computing," in *2012 6th International Conference on Application of Information and Communication Technologies (AICT)*, Oct 2012, pp. 1–3. x, 50, 51

[5] IBM, "Openafs administration guide." IBM Corporation, Feb 2016, pp. 6–7, 20, 40. [Online]. Available: http://docs.openafs.org/AdminGuide.pdf x, 31, 33, 35, 36, 37, 38, 43, 44, 45, 57, 58, 60, 63, 73

[6] M. Raynal, *Distributed Algorithms for Message-Passing Systems.* Springer Publishing Company, Incorporated, 2013. 1

[7] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Sistemas Distribuídos - Conceitos e Projeto*, 4th ed. Brasil: Addison-Wesley Publishing Company, 2007. 1, 2, 5, 11, 12, 15, 18, 44, 46

[8] G. Somasundaram and A. Shrivastava, *Information storage and management : storing, managing, and protecting digital information.* Indianapolis, IN, USA: Indianapolis, Ind. : Wiley Pub, 2009. 2, 4, 14, 20

[9] O. Kulyk, S. Neumann, J. Budurushi, and M. Volkamer, "Nothing comes for free: How much usability can you sacrifice for security?" *IEEE Security Privacy*, vol. 15, no. 3, pp. 24–29, 2017. 2

[10] A. Acquisti and J. Grossklags, "Losses, gains, and hyperbolic discounting: An experimental approach to information security attitudes and behavior," *UC Berkeley: 2nd Annual Workshop on "Economics and Information Security"*, p. 13, 04 2003. 2

[11] S. Parkin and S. Epili, "A technique for using employee perception of security to support usability diagnostics," in *2015 Workshop on Socio-Technical Aspects in Security and Trust*, July 2015, pp. 2–5. 2

[12] ISO, "ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability," International Organization for Standardization, Tech. Rep., 1998. [Online]. Available: http://www.userfocus.co.uk/resources/iso9241/part11.html 2

[13] F. Yahya, R. J. Walters, and G. B. Wills, "Analysing threats in cloud storage," in *2015 World Congress on Internet Security (WorldCIS)*, Oct 2015, pp. 44–48. 2

[14] A. Miede, N. Nedyalkov, C. Gottron, A. König, N. Repp, and R. Steinmetz, "A generic metamodel for it security attack modeling for distributed systems," in *2010 International Conference on Availability, Reliability and Security*, Feb 2010, p. 435. 2

[15] A. V. Uzunov and E. B. Fernandez, "An extensible pattern-based library and taxonomy of security threats for distributed systems," vol. 36, no. 4, 2014, pp. 734 – 747, security in Information Systems: Advances and new Challenges. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548913001827 3

[16] OpenAFS. (2018) Openafs website. [Online]. Available: https://www.openafs.org/ 3, 28

[17] G. Pék, L. Buttyán, and B. Bencsáth, "A Survey of Security Issues in Hardware Virtualization," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 40:1–40:34, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2480741.2480757 4

[18] "Information technology — security techniques — information security management systems — overview and vocabulary," vol. 2018, pp. 2, 5, 12, Feb. 2018. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/c073906_ISO_IEC_27000_2018_E.zip 4

[19] F. Milicchio and W. A. Gehrke, *Distributed Services with OpenAFS: For Enterprise and Education*, 1st ed. Springer Publishing Company, Incorporated, 2010. 5, 16, 18, 29, 32, 35, 38, 51

[20] R. S. Wazlawick, *Metodologia de Pesquisa para Ciência da Computação.* Elsevier, RJ, 2014. 7

[21] A. C. Gil, *Métodos e Técnicas de Pesquisa Social.* Atlas, RJ, Brasil, 1999, vol. 2008. 7, 8

[22] T. Gerhardt and D. Silveira, "Métodos de pesquisa." Plageder, p. 54. [Online]. Available: www.ufrgs.br/cursopgdr/downloadsSerie/derad005.pdf 8, 11, 53

[23] A. S. Tanenbaum and M. v. Steen, "Distributed systems: Principles and paradigms (2nd edition)." Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006, p. 590. 11, 12, 15, 16, 18, 20

[24] P. Bžoch and J. Šafařík, "Security and reliability of distributed file systems," in *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, vol. 2, Sept 2011, pp. 764–769. 12, 22, 23, 43, 77

[25] S. Mullender, Ed., *Distributed Systems (2Nd Ed.)*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1993. 12

[26] "Network Attached Storage | Glossário EMC." [Online]. Available: https://brazil.emc.com/corporate/glossary/network-attached-storage.htm 14

[27] S. Pawar, S. E. Rouayheb, and K. Ramchandran, "Securing dynamic distributed storage systems against eavesdropping and adversarial attacks," *IEEE Transactions on Information Theory*, vol. 57, no. 10, pp. 6734–6753, Oct 2011. 14, 23

[28] K. J. L. Kumar, "Implementing network file system protocol for highly available clustered applications on network attached storage," in *2013 5th International Conference and Computational Intelligence and Communication Networks*, Sept 2013, pp. 496–499. 14

[29] M. Delamaro, J. Maldonado, and M. Jino, *Introdução ao teste de software*. Rio de Janeiro: Elsevier, 2007. 16

[30] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982. 17

[31] K. Gupta and J. K. Saini, "Novel approach for distributed file system with multiple layers of fault tolerance," in *International Conference on Computing, Communication Automation*, May 2015, pp. 616–619. 17

[32] P. Ostovari and J. Wu, "Fault-tolerant and secure distributed data storage using random linear network coding," in *2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2016, pp. 1–8. 17, 22

[33] P. Bžoch and J. Šafařínk, "Simulation of client-side caching policies for distributed file systems," in *Eurocon 2013*, July 2013, pp. 679–686. 19, 34

[34] IBM, "2018 cost of a data breach study: Global overview," Tech. Rep. [Online]. Available: https://www.ibm.com/security/data-breach 20

[35] D. Modi, R. K. Agrawalla, and R. Moona, "Transcryptdfs: A secure distributed encrypting file system," in *International Congress on Ultra Modern Telecommunications and Control Systems*, Oct 2010, pp. 187–194. 20

[36] M. Grawinkel, M. Mardaus, T. Süß, and A. Brinkmann, "Evaluation of a hash-compress-encrypt pipeline for storage system applications," in *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Aug 2015, pp. 355–356. 21

[37] F. Graf and S. D. Wolthusen, "A capability-based transparent cryptographic file system," in *2005 International Conference on Cyberworlds (CW'05)*, Nov 2005, pp. 8 pp.–108. 21

[38] B. Choi, J. y. Sohn, S. W. Yoon, and J. Moon, "Secure clustered distributed storage against eavesdroppers," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6. 22

[39] Y. Tian, X. Qin, and Y. Jia, "Secure replica allocation in cloud storage systems with heterogeneous vulnerabilities," in *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Aug 2015, pp. 205–214. 23

[40] A. R. Yumerefendi and J. S. Chase, "Strong accountability for network storage," *Trans. Storage*, vol. 3, no. 3, Oct. 2007. [Online]. Available: http://doi.acm.org/10.1145/1288783.1288786 23

[41] G. Lizhong and J. Huibo, "Collaborative intrusion detection scheme for network-attached storage based on agents," in *2010 International Forum on Information Technology and Applications*, vol. 1, July 2010, pp. 444–447. 23

[42] M. Eikel, C. Scheideler, and A. Setzer, "Robust: A crash-failure-resistant distributed storage system," in *Principles of Distributed Systems - 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings*, 2014, pp. 107–122. [Online]. Available: https://doi.org/10.1007/978-3-319-14472-6_8 23

[43] H. Akutsu, K. Ueda, T. Chiba, T. Kawaguchi, and N. Shimozono, "Reliability and failure impact analysis of distributed storage systems with dynamic refuging," *IEICE Transactions*, vol. 99-D, no. 9, pp. 2259–2268, 2016. [Online]. Available: http://search.ieice.org/bin/summary.php?id=e99-d_9_2259 23

[44] R. H. Palacios, C. Rodríguez-Quintana, A. F. Díaz, M. Anguita, and J. Ortega, "Evaluation of redundant data storage in clusters based on multi-multicast and local storage," *The Journal of Supercomputing*, vol. 73, no. 1, pp. 576–590, 2017. [Online]. Available: https://doi.org/10.1007/s11227-016-1913-6 24

[45] W. Stallings, *Criptografia e segurança de redes: princípios e práticas, 6a. ed.* Pearson Education do Brasil, 2015. 24, 26, 66

[46] E. Rescorla, "The transport layer security (tls) protocol version 1.3," Internet Requests for Comments, RFC Editor, RFC 8446, August 2018. 25

[47] T. Ylonen and C. Lonvick, "The secure shell (ssh) protocol architecture," Internet Requests for Comments, RFC Editor, RFC 4251, January 2006, http://www.rfc-editor.org/rfc/rfc4251.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4251.txt 25

[48] A. S. Tanenbaum and M. v. Steen, "Distributed systems." Pearson, 2017, p. 596. [Online]. Available: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017/ 28, 30

[49] A. Elomari, A. Maizate, and L. Hassouni, "Data storage in big data context: A survey," pp. 1–4, Nov 2016. 29

[50] K. Bok, J. Lim, H. Oh, and J. Yoo, "An efficient cache management scheme for accessing small files in distributed file systems," in *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Feb 2017, pp. 151–155. 33

[51] IBM, "Openafs user guide." IBM Corporation, Feb 2000, pp. 6–7, 20, 40. [Online]. Available: http://docs.openafs.org/UserGuide.pdf 34, 35

[52] A. C. A.-D. Remzi H. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, May 2015. [Online]. Available: http://pages.cs.wisc.edu/~remzi/OSTEP/ 34

[53] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The kerberos network authentication service (v5)," Internet Requests for Comments, RFC Editor, RFC 4120, July 2005, http://www.rfc-editor.org/rfc/rfc4120.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4120.txt 39, 60

[54] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978. 39

[55] K. Raeburn, "Advanced encryption standard (aes) encryption for kerberos 5," Internet Requests for Comments, RFC Editor, RFC 3962, February 2005, http://www.rfc-editor.org/rfc/rfc3962.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3962.txt 39

[56] MIT. (2018) Rx - extended remote procedure call. [Online]. Available: http://web.mit.edu/kolya/afs/rx/rx.mss 42

[57] C. A. Latha and H. L. Shashidhara, "Clock synchronization in distributed systems," in *2010 5th International Conference on Industrial and Information Systems*, July 2010, pp. 475–480. 44

[58] I. Redbooks, *Auditing and Accounting on Aix*. Vervante, 2000. [Online]. Available: https://www.redbooks.ibm.com/redbooks/pdfs/sg246020.pdf 45

[59] IBM. (2012) The audit subsystem in aix. [Online]. Available: http://www-01.ibm.com/support/docview.wss?uid=isg3T1000212 45

[60] D. Noveck, P. Shivam, C. Lever, and B. Baker, "Network file system (nfs) version 4 protocol," Internet Requests for Comments, RFC Editor, RFC 7931, July 2016. 46

[61] H. W. Lim and G. Yang, "Authenticated key exchange protocols for parallel network file systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 92–105, Jan 2016. 46, 48

[62] G. Ruppert and P. L. de Geus, "Uma análise de segurança das versões do protocolo nfs," Tech. Rep., 2004. [Online]. Available: https://www.lasca.ic.unicamp.br/paulo/papers/2004-SSI-guilherme.ruppert-analise.seguranca.NFS.pdf 47, 48

[63] J. Cohen and S. Acharya, "Towards a more secure apache hadoop hdfs infrastructure," in *International Conference on Network and System Security*. Springer, 2013, pp. 735–741. 51

[64] O. O'Malley. (2018) Motivations for apache hadoop security. [Online]. Available: https://br.hortonworks.com/blog/motivations-for-apache-hadoop-security/ 51

[65] Apache. (2018) Apache hadoop project. [Online]. Available: http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html 51

[66] J. E. Mota Filho, *Análise de Tráfego em Redes TCP/IP: Utilize tcpdump na análise de tráfegos em qualquer sistema operacional*. Novatec Editora, 2013. 61, 62

[67] N. Williams, "A pseudo-random function (prf) for the kerberos v generic security service application program interface (gss-api) mechanism," Internet Requests for Comments, RFC Editor, RFC 4402, February 2006. 73

[68] P. Honeyman, L. Huston, and M. Stolarchuk, "Hijacking afs," Center for Information Technology Integration, Tech. Rep., 1991. 74

[69] N. W. Group. (2019) rxgk: Gssapi based security class for rx. [Online]. Available: https://tools.ietf.org/id/draft-wilkinson-afs3-rxgk-04.html#rfc.section.8.4 74, 75

[70] MIT. (2019) Encryption types. [Online]. Available: https://web.mit.edu/kerberos/kfw-4.1/kfw-4.1/kfw-4.1-help/html/encryption_types.htm 78

[71] M. Wagner, S. Heyse, and C. Guillemet, "Brute-force search strategies for single-trace and few-traces template attacks on the des round keys of a recent smart card." *IACR Cryptology ePrint Archive*, vol. 2017, p. 614, 2017. 78

[72] N. Gupta, D. Chang, S. K. Sanadhya, and S. Mukhopadhyay, "Side channel collision attack on twine-80 and des with reduced masked rounds," Ph.D. dissertation, 2015. 78

[73] M. Sharma and R. Garg, "Des: The oldest symmetric block key encryption algorithm," in *2016 International Conference System Modeling & Advancement in Research Trends (SMART)*. IEEE, 2016, pp. 53–58. 78

[74] A. Chernyakhovsky, "Improving The OpenAFS Security Model Without Client-side Changes." [Online]. Available: http://web.mit.edu/achernya/Public/thesis.pdf 79

[75] I. Plotnik, T. A. Be'ery, M. Dolinsky, O. Plotnik, G. Messerman, and S. Krigsman, "System, method and process for detecting advanced and targeted attacks with the recoupling of kerberos authentication and authorization," Mar. 3 2016, uS Patent App. 14/474,198. 83

[76] F. E. Catota, M. G. Morgan, and D. C. Sicker, "Cybersecurity incident response capabilities in the ecuadorian financial sector," *Journal of Cybersecurity*, vol. 4, no. 1, p. tyy002, 2018. [Online]. Available: http://dx.doi.org/10.1093/cybsec/tyy002 84